INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Fractal Engine:

An Affine Video Processor Core for Multimedia Applications

Omid Fatemi

A Dissertation

Submitted to the School of Graduate Studies and Research

in fulfillment of the requirements

for the Degree of

Ph.D. in Electrical and Computer Engineering

Ottawa-Carleton Institute of Electrical Engineering

Department of Electrical Engineering

School of Information Technology and Engineering

University of Ottawa

September, 1999

بسم ا... الرحمن رحيم

In the Name of Allah

To Mahdi (AJ),

Acknowledgements

All praise is due to Allah, the Lord of the Worlds, the Beneficent, the Merciful. Who is the creator, who taught with the pen and who taught man what he knew not. And my thanks to his representative in this world, the Imam of the time, Mahdi (AJ) who is the source of all favors and knowledge.

It is my pleasure to acknowledge and thank all persons who have influenced me in the course of this research. First, I express my gratitude to my supervisor Dr. Sethuraman Panchanathan for introducing me to the exciting fields of multimedia, video and image processing, parallel processing and VLSI and for his continued support and encouragement during my thesis work. I would also like to express my gratitude to my co-supervisor Dr. Sunil R. Das for his continuos support.

I would like to thank all the past and present members of the Visual Computing and Communications Laboratory, especially Mahmoud Reza Hashemi for his help and cooperation.

My special thanks are due to all the support staff members of School of Information Technology and Engineering for their help, especially Michele Roy and Lucette Lepage.

The generous financial support of Ministry of Culture and Higher Education of the Islamic Republic of Iran and NSERC that made this research possible is also gratefully acknowledged.

My beloved wife showed constant understanding and support. A mere acknowledgment by no means compensates the hardship she had to go through on the account of my research work.

Last, but not least, special thanks to my dear parents who always supported me in the wonderful world of science and encouraged me in getting over all the difficulties.

Fractal Engine

An Affine Video Processor Core for Multimedia Applications

Abstract

The recent advances in VLSI technology, high-speed processor designs, Internet/Intranet implementations, broadband networks (ATM and ISDN) and compression standards are leading to the popularity of multimedia applications. In general, multimedia computing presents challenges from the perspectives of both hardware and software. Each medium in a multimedia environment requires different processes, techniques, algorithms and hardware. Hence, it is crucial to design a generic processor architecture that meets the computing requirements of the various media types. In another word, there is a need for a bottom-up design strategy for meeting the computing needs of multimedia processing.

In this thesis, we propose the design of an affine video processor termed Fractal Engine. We have first derived the fundamental operations involved in visual processing tasks and designed the generic processing elements to map a majority of these operations. We have chosen affine transformations as the target algorithm as it is expected to be increasingly used in many visual-processing applications including latest video coding standard MPEG4. We have chosen fractal block processing (FBP) as a candidate algorithm for the

design of target video processor, since it encompasses a variety of visual processing operations including affine transforms.

Fractal Engine is capable of implementing the gamut of image/video processing algorithms. Fractal Engine is a simple, modular, and scalable architecture that is optimized to execute both low-level and mid-level operations. It is capable of implementing a variety of visual processing tasks. Fractal Engine is an open architecture and is therefore capable of adapting to the processing requirements of a variety of media processing algorithms. The individual modules of the Fractal Engine have been implemented in VHDL. A behavioral model of the circuit has been developed and fully tested by using VHDL simulators. The model is synthesized using BiCMOS .8μ ASIC library cells and Xilinx/Altera FPGAs. We have chosen to demonstrate the real-time execution capability of Fractal Engine by mapping specific visual processing algorithms such as fractal block coding (FBC), vector quantization and motion estimation onto the proposed architecture.

(Blank Page)

Table of Contents

Table of Figures

1 Introduction

The recent advances in VLSI technology[1]-[2], high-speed processor designs[3], Internet/Intranet implementations[4], broadband networks[5] (ATM and ISDN) and compression standards [6] (JPEG [7], MPEG [8], H.261, H.263 and G.273) are leading to the popularity of multimedia applications. Example applications include Multimedia Information Systems [9], Digital Libraries [10], Remote Sensing and Natural Resources Management [11] and Geographic Information System [12].

A variety of media processing techniques are typically used in multimedia processing environments to capture, store, manipulate and transmit multimedia objects such as text, handwritten data, audio objects, still images, 2D/3D graphics, animation and full-motion video. Example techniques include speech analysis and synthesis, character recognition, audio compression, graphics animation, 3D rendering, image enhancement and restoration, image/video analysis and editing, and video transmission.

Visual media in a multimedia system contains a significant amount of information, and correspondingly involves a large volume of data in contrast to the other media types. Uncompressed digital video requires 250 Mb/s to support studio quality transmission of NTSC images (480 lines x 720 pixels/line x 24 bits/pixel x 30 frames/s)[13]. Even a simpler application such as video telephony (240 lines x 360 pixel/line x 16 bits/pixel x 10) requires 14 Mb/s to transmit the digital video signal in raw format. The bandwidth and storage requirements of visual information typically make it difficult to manage the data in its raw form. However, there is considerable redundancy in video data, both from

an information theoretic viewpoint as well as from the perspectives of structural content and human perception. A number of image and video compression standards, e.g., MPEG-1[14], MPEG-2[15], MPEG-4[16],[17], H.261[18], and H.263[19] have been recently proposed to compress the visual data for a variety of transmission and/or storage applications. There is ongoing research and standardization efforts targeted towards future multimedia applications with the objective of integrating compression and content access functionality, including MPEG-7[20]. These techniques and standards will involve execution of complex video processing tasks in real-time. The challenges can range from waveform coding implementations to scene modeling and understanding. For example, the principal objective of model-based image coding [21], [22] or intelligent image coding is to understand the scene by modeling the objects in order to achieve a higher level representation. In addition, there is an increasing interest in 3-D image and video processing[23], [24]. An important processing task in most of these situations is affine transformation[25], which includes operations such as rotation, transposition, scaling and translation. For example, intelligent motion estimation in a video sequence requires extraction of the motion of objects and camera operations, which could be represented using affine parameters.

In general, multimedia computing presents challenges from the perspectives of both hardware and software. Each media in a multimedia environment requires different processes, techniques, algorithms and hardware. Hence, it is crucial to design a generic processor architecture that meets the computing requirements of the various media types. The complexity, variety of techniques and tools, and the high computation, storage and

I/O bandwidths associated with visual processing pose several challenges, particularly from the viewpoint of real-time implementation.

Real-time implementation goal is the principal reason for the slant of most media processor development[26] towards visual processing. Several processing solutions ranging from multimedia extensions to general purpose processors such as the Intel MMX[27], programmable DSP architectures such as the TI-MVP [28], Media processors like the Philips TriMedia processor[29], and special purpose architectures such as the C-Cube MPEG decoder chip-sets[30] have been proposed to implement a variety of multimedia (particularly visual) processing operations. A detailed categorization of available multimedia processing strategies is required in order to propose the optimum architecture for target applications. In this thesis, we have designed a high performance visual signal processor (VSP) called Fractal Engine, which is optimized to execute a variety of both mid-level and low-level visual operations.

1.1 Motivation

The implementation of video processing algorithms or in general multimedia algorithms demands systems of large computational capability with efficient VLSI implementation of the various media processing algorithms. Real time video compression requires processing power in the range of 100 MOPS to 100000 MOPS. The envisaged mass application of digital multimedia demands fast and reduced size implementations, which are potentially feasible due to recent advances in VLSI technology[31] specifically in the areas of high density, and fast circuit implementations. VLSI technologies have now advanced to the point where, for some applications, the processing power and memory

required to perform these tasks can be incorporated into a few silicon chips. Individual transistors switch faster and therefore circuits perform operations at a higher speed. The transistors occupy less space and therefore more complicated design can be integrated into a single chip. It is required to study various options in VLSI design and select the best environment for target applications.

The advent of hardware description languages such as VHDL (VHSIC Hardware Description Language)[32] and re-configurable high density FPGAs[33] (Field Programmable Gate Array) such as Altera[34] and Xilinx[35] have not only facilitated rapid prototyping of digital designs, but also serve the needs for programmable and re-configurable hardware design. Thus it makes possible quick assembly of pre-designed generic processing elements into architectures that can be dedicated to execute specific algorithms or a class of algorithms under the assumption that the generic processing elements were designed to accommodate a variety of media processing requirements. A specific configuration can also be chosen from a universal architecture using external control signals assuming that the target processor is capable of organizing the generic processing elements into various configurations. Hence, enabling VLSI technology should be thoroughly studied and the best possible combinations of HDLs, ASICs (Application Specific Integrated Circuits) and FPGAs have to be selected.

The main focus of the researchers for video processor design is the optimization of low-level operations such as multiplication and accumulation. However, these developments are not sufficient to overcome all the problems in implementing multimedia applications. There is clearly a need for a bottom-up design strategy for meeting the computing needs

of multimedia processing. We need to derive the basic operations involved in a variety of image and video processing operations such as enhancement, restoration, compression and analysis of images and video sequences. It is required to derive mid-level and high-level operations in visual domain and design scalable and modular architectures for these requirements. Since Fractal Block Processing (FBP)[36] encompasses a majority of image processing operations[37],[38], including summation/accumulation, image addition / subtraction, translation, stretching, shifting, scaling, rotation and pattern matching, we have chosen this as the ***candidate algorithm*** for the design of the generic video processing element. We note that the operations of translation, stretching, shifting, scaling and rotation termed as affine transforms[39] are particularly important and are extremely powerful in visual processing tasks such as image analysis and understanding motion in video. It is our belief that most of the complex processing operations involved in the next generation of visual processing tasks will involve some form of affine transformation. We note that there is hardly any architectural solutions that emphasize affine transform implementation in the context of general purpose video processing. The choice of Fractal block processing as the candidate algorithm in our generic processing element design is therefore based on the following two premises: (i) it contains a reasonable super-set of the variety of processing tasks (including affine transformations) typically found in visual (and multimedia) processing, and (ii) it is a computationally intensive procedure and hence presents challenges from the perspective of real-time implementation. Another important requirement in the design of multimedia processor architectures is scalability. For example, visual processing tasks typically operate on a variety of image sizes, resolutions, and frame rates, and it is therefore crucial to design

the generic processing element to be scalable. For a problem of complexity X which is executed using N units in T seconds, scalability implies the following: (i) T/M seconds will be required to solve the problem using NM units, and/or (ii) A problem of complexity XM is solved in T seconds using NM units. The first type of scalability requires a flexible control design while the second type of scalability requires that the feature of scalability be incorporated in the design of individual modules.

An important factor in designing a high performance video processor is to adopt the promising features in existing architectures. This necessitates full investigation of a variety of existing processors ranging from general purpose processors to dedicated hardware modules used in multimedia applications.

1.2 Problem Statement

In this thesis, we propose the design of generic processing elements based on the derivation of the fundamental visual processing operations in Fractal block processing. An Affine Transform Processor (ATP), which is the core processor, and further a visual signal processor based on ATP core are designed. The processor termed Fractal Engine is capable of implementing the gamut of image/video processing algorithms. Fractal Engine is a simple, modular, and scalable architecture that is optimized to execute both low-level and mid-level operations. It is capable of implementing a variety of visual processing tasks, including spatial filtering, contrast enhancement, frequency domain operations, histogram calculation, geometric transforms, indexing, vector quantization, fractal block coding, shot boundary detection, motion estimation, and camera operation detection.

Fractal Engine is an open architecture and is therefore capable of adapting to the processing requirements of a variety of media processing algorithms. The individual modules of the Fractal Engine have been implemented in VHDL. A behavioral model of the circuit has been developed and fully tested by using VHDL simulators. The model is synthesized using BiCMOS .8μ ASIC library cells[40] and Xilinx/Altera FPGAs. We have chosen to demonstrate the real-time execution capability of Fractal Engine by mapping specific visual processing algorithms such as fractal block coding (FBC)[41], vector quantization[42] and motion estimation[43] onto the proposed architecture. The steps adopted in the design of the Fractal Engine are presented in Figure 1.

1.3 Outline of the Thesis

The thesis is organized as follows. Chapter 2 presents the fundamental operations in visual media processing. All of the visual algorithms are classified to four major categories and six different classes. The basic operations of all groups are then introduced. We propose that a general affine transformation is a mid-level fundamental operation which involves low-level operations in image/video processing algorithms. MPEG4 is used as an example to demonstrate the validity of our categorization. In chapter 3, different aspects of enabling VLSI technology are reviewed. The different options for VLSI implementation of video signal processors are then discussed. Hardware description languages, logic synthesizers, and behavior compilers for multimedia purposes are then explained and the necessary tools in our design methodology are introduced. The design trends in multimedia processor architectures are detailed in chapter 4.

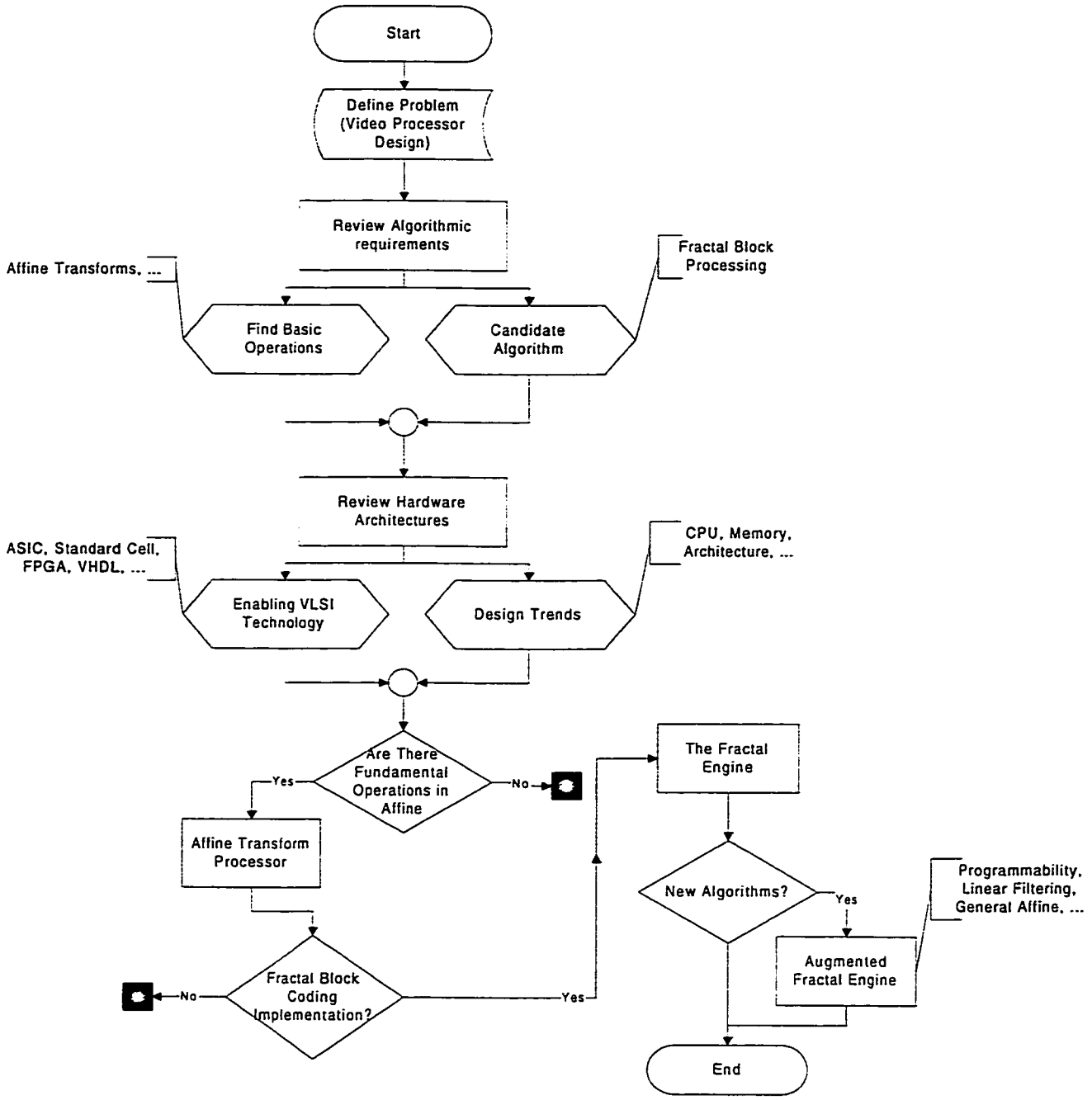


Figure 1 - The design approach for the Fractal Engine

It is concluded that dedicated modules are suitable for critical units while programmable modules are required to facilitate the adaptation of the architecture for various algorithms. The individual modules of the proposed ATP core are presented in chapter 5, where basic

operations in affine transformation are introduced and an optimal scalable architecture is proposed. The detailed design of the Fractal Engine which is optimized for executing fractal block processing algorithm is presented in chapter 6. The mapping of example algorithms onto the Fractal Engine and area/timing analysis are also discussed in this chapter. Since Fractal Engine is an open architecture, we have demonstrated the implementation of new algorithms such as generalized affine transform operations and interpolation filter in chapter 7. The design is based on the augmentation in terms of flexibility and programmability in Fractal Engine. Finally, conclusions and directions for future work are presented in Chapter 8 followed by the references.

2 Fundamental Multimedia Operations

We note that a critical analysis of image and video tasks will result in the derivation of the set of generic operations, which are typically employed in a variety of multimedia applications. In this chapter, we first present the fundamentals of visual media processing. We then summarize the various categories of operations in image and video processing followed by the derivation of the generic operations for visual processing and a brief introduction to affine transforms and fractal processing. The last section introduces MPEG4, the new standard for multimedia applications. The principal objective is to demonstrate that the candidate algorithm chosen for the design of the target architecture encompasses a majority of the visual operations as well as presents challenges from the perspective of real-time implementation.

2.1 Visual Media Basics

Video sequences are essentially a collection of individual frames (images). Hence, the main part of this section deals with the definitions for digital images.

Image Components

A digital image is composed of discrete points with a quantized value assigned to each point. In the case of gray-scale images this value represents the gray level of the point. However, for color images, the quantized value represents the color component values of the point.

A digital image is created from a continuos-tone image after the two steps of sampling and quantization[44]. In the sampling process, the brightness values of particular positions are sampled. In the quantization process, the sampled value is quantized to a fixed length integer value which is usually 8-bits for gray-scale images and 24-bits for color images. The 24-bit quantization known as true color representation consists of 3 independent 8-bit integer values describing the intensity of basic colors red, green and blue. This representation is known as RGB format. In the case of gray scale images, this value describes only the intensity value corresponding to the brightness of the point. A quantized sample representing the brightness value for a specific position in the image is called a *pixel* or a picture element. The combination of sampling and quantization processes is referred to as *image digitization.*

An image is generally sampled into a rectangular array of pixels. Each pixel has an (x,y) coordinate which describes its location in the image. The x-axis is the horizontal axis from left to right while the y-axis is the vertical axis from top to bottom as shown in Figure 2. The origin or location *(0,0)* is in the upper left corner of the image in this representation. As an example, the pixel at location *(50, 120)* is marked in the Figure 2.

The number of columns or rows in an image (M and N respectively for an MxN image) indicates the *spatial resolution* of the image which is directly related to the quality of the image. Spatial resolution describes how many pixels are in the image. The more pixels in the image, the better its quality and the larger its storage size. The number of pixels in a digital image depends on how finely the image is sampled, or divided into discrete pixels.

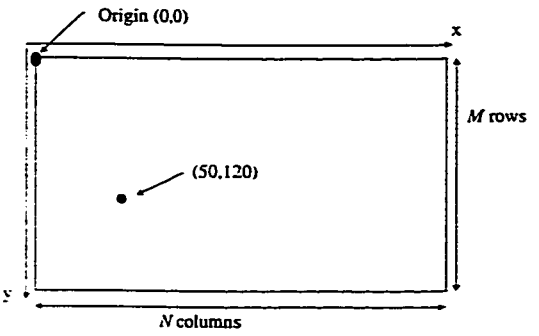


Figure 2- Image Coordinates

It is the sampling rate, which determines the number of pixels for a known physical size. For example, 200 dpi (dot per inch) means that there are 200 pixels in an inch. The maximum sampling rate is set by the digitizing device such as scanner, digital camera, etc. From sampling theorem, the necessary sampling rate so that the digital image adequately resolves all spatial details of the original continuous-tone image, is at least twice as fast as the highest spatial frequency contained in the image (Nyquist rate).

If sampling occurs at a lower rate than that required by the sampling theorem, the higher spatial frequency details will be lost in the digital image. Hence, the digital image will appear to be not as sharp as the original image. In Figure 3, four different sampling rates are employed to generate the illustrated digital pictures. It is clear that the picture (D) doesn't contain all the details of the picture (A) and the details of the picture are lost due to pixel *blocking* effect.

On the other hand, if sampling occurs at a higher rate than Nyquist rate, extra pixels will be created which theoretically do not contribute to improving image quality. However, they can be used in future manipulations such as image resampling/interpolation and feature extraction.

A) 96 dpi B) 72 dpi

C) 48 dpi D) 24 dpi

Figure 3- Barbara image in 4 different sampling rates.

Frame Rate

This parameter which is employed in video sequences determines the temporal resolution while the sampling rate determines the spatial resolution. The higher the frame rate, the more accurate is the motion representation in a digitized video sequence.

Frame rate is a sampling terminology, which is applied to digital and other non-digital forms of sequential image acquisition and display such as broadcast television. It is often expressed as the number of frames per second (fps). For example, if the frame rate of a system is 30 frames per second, an image frame is acquired (or displayed) every $1/30^{th}$ of a second. If an object being imaged moves across the image frame at a faster rate, it may never be captured in an individual image. Once again, we note that it is essential that the frame rate is at least twice as fast as the highest frequency of motion in the video sequence.

Interlaced Versus Progressive Scan

The concept of frame rate for image display relates how often an image is updated on the viewing display. Since the normal display mechanism is a video display monitor, images must be repeatedly refreshed. The rate at which images are refreshed can cause display flicker, and therefore human eye fatigue. Display flicker also depends on how the image is scanned on the display monitor. Common broadcast television equipment uses a technique known as *interlaced* scan[45] display, as shown in Figure 4. This means that the odd-numbered lines of the image are displayed first, followed by the even-numbered lines. The effect is to interleave, in time, the two interlaced halves of the image, one after another. Interlacing gives the impression to the observer that a new frame is present twice as often as it really is. This technique was used originally for television broadcast signals since the display could be refreshed less frequently without noticeable image flickering (although some minor line-to-line flicker does occur at certain instances). Systems using a standard commercial broadcast television display monitor for image display typically have a 30 frame per second frame rate and interlaced scan.

In motion image sequences, interlaced scan displays can show noticeable motion defects because the odd and even halves of each image are separated in time by one-half the frame rate. The result is a tearing effect that appears on objects with fast motion through the image frame.

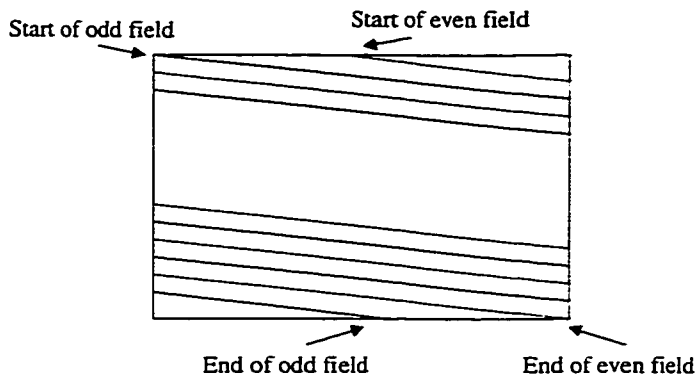


Figure 4- Interlaced scan display.

In the non-interlaced method known as *progressive* method[46], the entire image is displayed in one pass. In this case, the frame rate must be twice that of an equivalent interlaced display, or image flickering will be noticeable. Progressive scan eliminates line-to-line flicker and motion artifacts in displayed images. Systems using a progressive scan display monitor for image display typically have a 72 frame per second frame rate.

Fidelity Criteria of Digital Images and Video Sequences

Fidelity criteria are applied to measure image quality and for comparing the performance of different processing techniques. There are two types of criteria[38] that are used for evaluation of image quality, subjective and quantitative. The subjective criteria use human feeling about an image (or video sequence). Quantitative measures, try to describe or compare the image/video quality by an analytic formula.

Mean square criterion is often used in image processing. It refers to the average (or sum) of squares of the error between two images (u and u') and it could be described in three different formats as follow:

Average least square: $\sigma_{ls}^2 = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}|u(m,n)-u'(m,n)|^2$

Mean square error: $\sigma_{ms}^2 = \frac{1}{MN}E[|u(m,n)-u'(m,n)|^2]$

Average mean square: $\sigma_a^2 = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}E[|u(m,n)-u'(m,n)|^2]$

where the size of the image is *MxN* and *E[]* represents the mathematical expectation.

In many applications the mean square error is expressed in terms of a signal-to-noise ratio (SNR), which is defined in decibels (dB) as follows:

$$SNR = 10\log_{10}\sigma^2/\sigma_e^2 \qquad \sigma_e = \sigma_a, \sigma_{ms}, \sigma_{ls}$$

where σ^2 is the variance of the original image.

The alternative formula for SNR, called peak-to-peak SNR (PSNR) is defined as:

$$PSNR = 10\log_{10}\frac{(\text{peak - to - peak value of the reference image})^2}{\sigma_e^2} \qquad \sigma_e = \sigma_a, \sigma_{ms}, \sigma_{ls}$$

2.2 Digital Image and Video Processing Categories

Visual media processing involves operations to enhance, restore, compress and analyze images and video sequences. Whatever the operation, a similar set of steps are followed: A digital technique is applied to a digital image or video to form a digital result, such as a processed image/video, a compressed bit-stream or a list of extracted features. The four main categories of image/video processing tasks are now presented.

Enhancement:

The quality enhancement is the primary goal in digital signal processing systems. Many enhancement techniques are introduced to compensate for the effects of a specific (known or estimated) degradation process for 2-D signals known as images. This approach, known as image restoration, will be discussed in the next section. In image enhancement methods, little or no attempt is made to estimate the actual degradation process that has occurred on the picture. These include methods of modifying the intensity value, contrast enhancement, edge enhancement, deblurring, and smoothing or removing noise. These methods assume that certain general properties of the picture are degraded and attempt to resolve these problems. For example, increasing the contrast is a reasonable enhancement operation due to the attenuation of the picture, or deblurring is reasonable as shown in Figure 5 because degradation usually blurs and smoothing is justifiable, since noise is generally added to the original picture as shown in Figure 6.

Figure 5- Sharpening effect

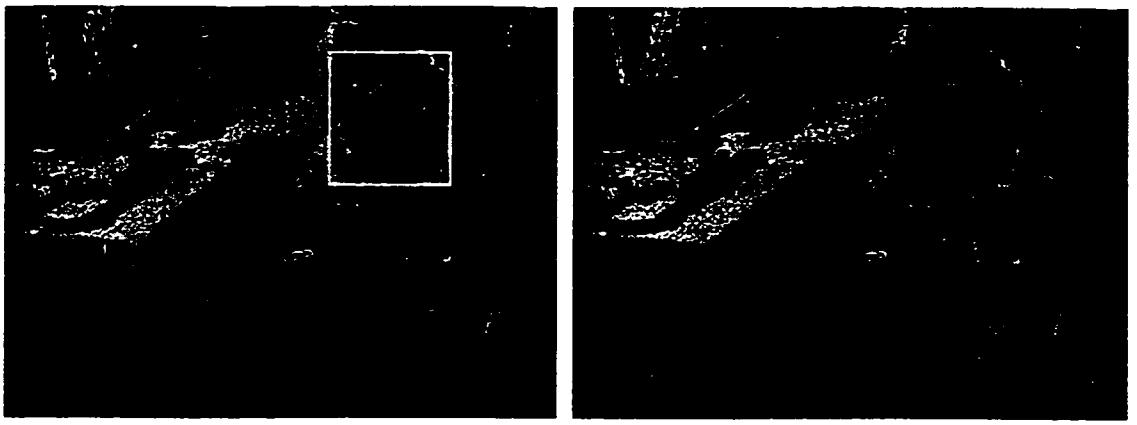


Figure 6- Noise removal using image enhancement techniques.

Image enhancement employs local correlation between adjacent pixels to enhance the image quality.

Restoration:

Picture restoration is applied on images that have been degraded in the presence of one or more sources of corruption. There are different kinds of degradations based on the affected area. *Point degradations* only alter the intensity value or the color of individual pixels while *spatial degradations* blur an area of the image. Other types generate *temporal degradation* in video sequences. For example, the pictures obtained in remote sensing and astronomy are degraded by atmospheric turbulence, aberrations of the optical system and relative motion between the camera and the object. In image restoration, it is assumed that the degraded image is a convolved version of the original image by the degradation function plus additive noise. The goal is to obtain as good an estimate as possible of the original picture. Obviously, any such estimation procedure requires some form of knowledge concerning the degradation function. As a result, frequency domain correlation is used to restore visual information. Examples include inverse filtering, pseudo inverse filtering, etc.

Compression:

The aim of digital data compression is to represent the data by as few bits as possible for the purpose of transmission or storage. The bandwidth and storage requirements of visual data typically make it impossible to handle visual (digital) data in its raw form and hence, a number of compression techniques have been developed.

Visual data compression methods fall into two common categories. In the first kind, called *lossless* compression, the data could be restored completely after the compression process. In this method, the redundancy of the image is exploited using source-coding techniques[101] such as Huffman coding and arithmetic coding. In the second category, called *lossy* techniques, higher compression rate is achieved at the expense of loosing some insignificant information in the decoding stage. This compression technique results in some distortion. Efficient compression techniques tend to minimize the distortion perceived by human visual system.

Different image and video compression techniques[74] remove the existing redundancy in different domains and hence, can be classified as follows:

Spatial based

In this class of compression techniques, the existing correlation within an image such as predictability, randomness and smoothness is exploited. ADPCM (adaptive differential pulse code modulation), vector quantization and fractal block coding techniques are typical examples of this category.

Temporal based

In this category, the existing correlation within a video sequence and between the consecutive frames of the same shot is exploited and the redundancy is removed. Motion estimation[95],[99] is the basic operation for these techniques.

Frequency based or transform coding

In transform coding techniques, a block of data is transformed so that a large fraction of its total energy is condensed into a small part of the transformed data which are quantized independently. DCT (discrete cosine transform)[75] and DWT (discrete wavelet transform)[94],[97] are typical digital transforms employed in this technique.

We note that the objective in all categories is to exploit the spatio-temporal correlation in an image or video to reduce the redundancy and represent the data in a compressed form.

Image/Video Analysis:

Semantic correlation of the pixels is used for image and video understanding. Recently, there is a tendency to represent multimedia objects using general object based representations which provides content-based functionalities. The objective of model-based image representation[21] or intelligent image understanding is to understand the scene by modeling the objects, yielding a higher level representation. Applications of model based image representation and image analysis include, automatic vehicle driving, medical inspection, mobile robot navigation, mail sorting, label reading, global model construction and low-bandwidth image coding. Semantic correlation of the pixels is used for image and video understanding. Example operations involved in image and video

analysis include, image segmentation, feature extraction, object classification, indexing, scene cut detection, etc.

It can be deduced from the summary of the different categories of operations listed above that the principal task is to exploit the different forms of correlations present in the visual data. The individual operations encountered in visual processing are detailed in the following section.

2.3 Fundamental Operations in Visual Processing

The fundamental operations in the four major categories of visual processing tasks are listed below. The objective is to select the candidate algorithm, which will be employed in the design of a high performance video processor. The selected kernel algorithm needs to be represented by a majority of such operations. We now propose the categorization of all individual operations in six classes as shown in Figure 7. The individual operations of each class are detailed.

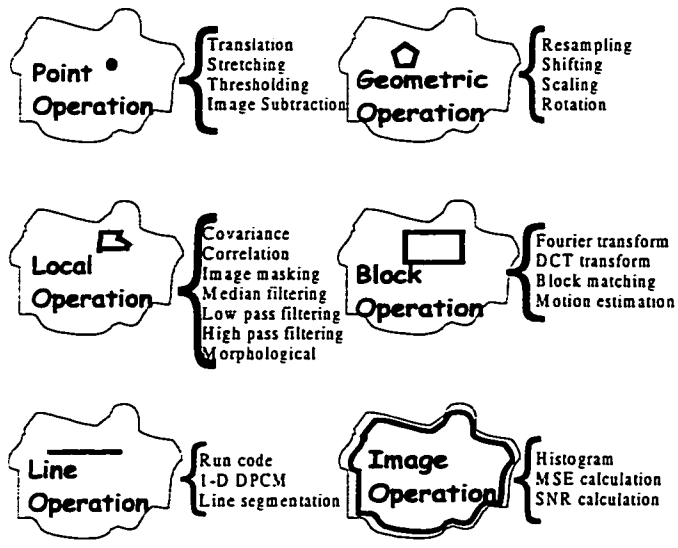


Figure 7- Visual Media Operations

Point operations

The resulting gray level at a pixel depends only on the input gray level of one point (usually the gray level of the same point before applying the operation). Such operations are used for gray scale manipulation and for segmentation by pixel classification. The extension of these operations include dual image operations where the output level of a pixel depends only on the set of input levels from the pixels at the same position. For example, we may want to take the difference between two pictures. The operations in this category are :

Translation

Description: Each image pixel is added to a constant translation factor.

$$O(x,y) = I(x,y) + t_f$$

Where $O(x,y)$ is the output image, $I(x,y)$ is the input image and t_f is the translation factor.

Applications: Brightening or darkening the image (an example is shown in Figure 8-B).

Category: Single-image.

Stretching

Description: Each image pixel is multiplied by a constant stretch factor (s_f).

$$O(x,y) = I(x,y) * s_f$$

Applications: Increasing or decreasing the contrast of the image (an example is shown in Figure 8-C).

Category: Single-image.

Thresholding

Description: Each image pixel is evaluated to be above or below a predetermined threshold value (t_v). If the pixel brightness is less than the threshold, the resulting pixel brightness is set to 0, otherwise it is set to the maximum value (e.g. 255 for 8 bit values).

$$O(x,y) = 0 \qquad if\ I(x,y) < t_v$$

$$O(x,y) = 255 \quad if\ I(x,y) \geq t_v$$

Applications: Creating a very high contrast image, segmenting the image by highlighting an object of interest and separating it from its background (an example is shown in Figure 8-D).

Category: Single-image.

Image Subtraction / Addition

Description: One image is subtracted from or added on a pixel by pixel basis to a second image.

$$O(x,y) = I_1(x,y) \pm I_2(x,y)$$

Applications: Removing common background image information, determining object motion, Averaging over images of the same scene to reduce random noise, merging two images.

Category: Dual-image.

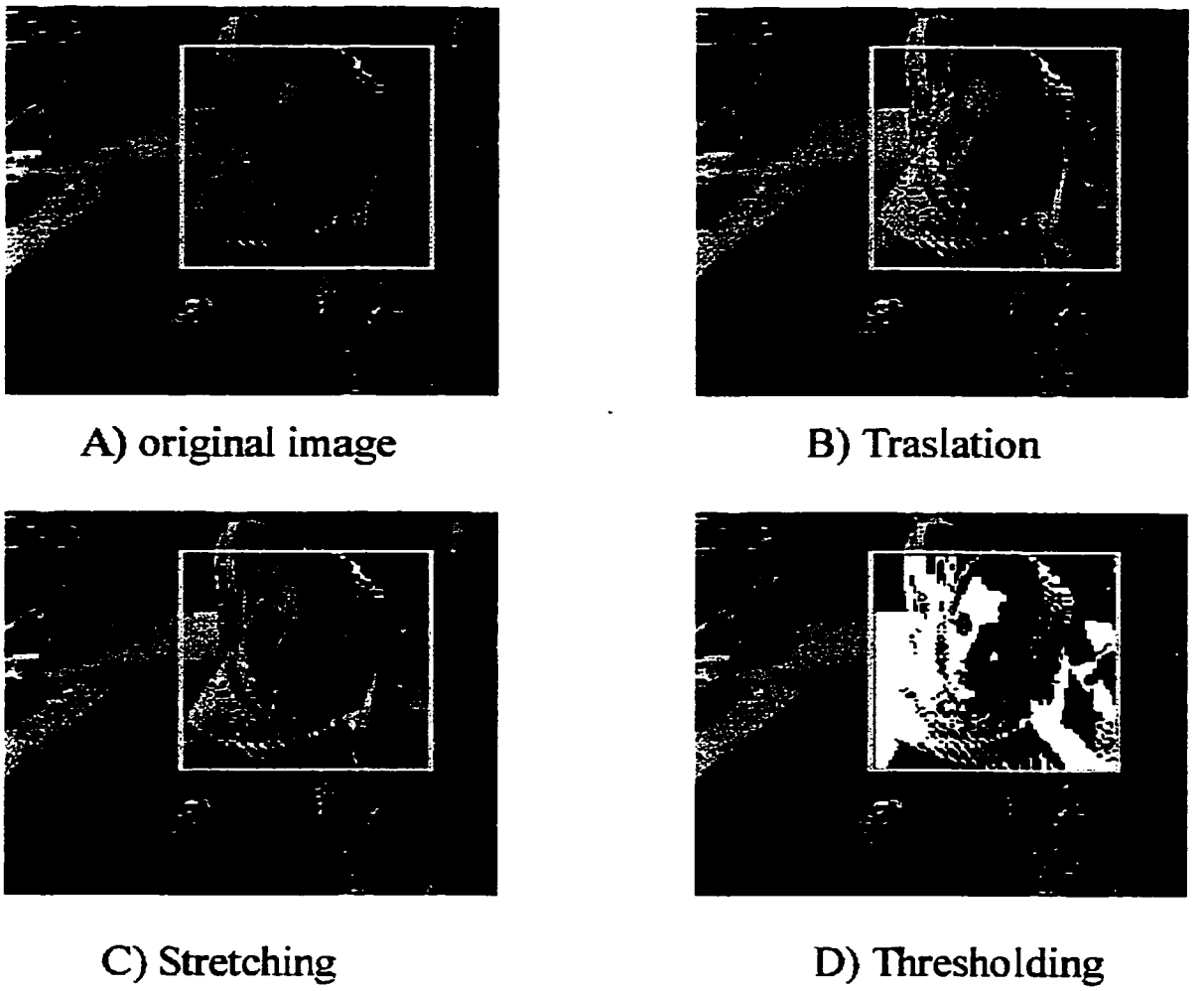


Figure 8- Example point operations

Local operations

The output of these operations depends only on the gray values in a neighborhood of a particular pixel. Such operations are used for noise cleaning, edge and local feature detection, etc. The following operations belong to this category:

Image Masking

Description: A finite impulse response (FIR) filter or mask is applied to the image to perform a spatial image processing operation.

$$O = m * I$$

where * indicates masking operation.

Applications: Low-pass filtering, unsharp masking, high-pass filtering, edge enhancement and line detection.

Category: dual-image.

Median Filtering

Description: The filter is a ranking filter, where for example the fifth-ranked pixel brightness value is selected as the output pixel brightness from a 3x3 input group of pixels.

$$O(x,y) = f(I)$$

where f is the nonlinear ranking and selecting function and I is the input group of pixels.

Applications: Removing impulse noise spikes from an image.

Category: single-image.

Morphological operations such as erosion and dilation

Description: The erosion operation reduces the size of bright objects on a dark background in an image and the dilation operation increases the size of bright objects on a dark background in an image (morphological process).

Applications: Image analysis, outlining, thinning, skeletonization and edge detection.

Category: single-image.

Line operations

The inputs to these operations are pixel values which reside across a vertical or horizontal line. Such operations are typically used in raster scan images. A typical example operation is the grouping of one's and zero's in a line for run-length coding. Example operations in this category include:

Run Length Coding

Description: Across each line of an image, pixel values are sequentially compared and grouped together into runs of identical brightness.

Applications: image compression.

Category: Single image.

Differential Pulse-Code Modulation (DPCM) Coding

Description: Each pixel value is replaced by the difference value of it and its neighbor and then represented by a lower-resolution value.

Applications: Lossy image compression.

Category: single-image.

Line segmentation

Description: Each line in an image is scanned and the white intervals are recognized to segment each line from the image.

Application: Text recognition.

Category: single-image.

Geometric *operations*

These operations are performed on a set of pixels defined by a geometrical transformation or around a neighborhood of a specified point. This class includes the following operations:

Up and Down Sampling

Description: Portion of image is resampled for another spatial resolution.

Applications: Image enhancement, zooming in and out, and image size adjustment.

Category: single-image.

Shifting

Description: The spatial location of image pixels is shifted linearly.

$I(x,y) \rightarrow O(x',y')$ *where* $x'=x+T_x$ *and* $y'=y+T_y$

Applications: Geometric adjustment of the location of an image.

Category: single-image.

Scaling

Description: The spatial size of image pixels is expanded or reduced linearly.

$I(x,y) \rightarrow O(x',y')$ *where* $x'=x.S_x$ *and* $y'=y.S_y$

Applications: Geometric adjustment of the size of an image.

Category: single-image.

Rotation

Description: The image is rotated linearly about the origin.

$I(x,y) \rightarrow O(x',y')$ *where* $x'=x.\cos(\theta)+y.\sin(\theta)$ *and* $y'= -x.\sin(\theta)+y.\cos(\theta)$

Applications: Geometric adjustment of an image.

Category: single-image.

Block operations

A rectangular block of pixels with a typical size of *4x4*, *8x8* or *16x16* are grouped together and processed. These operations may result in another block, a single value or a vector of data. Example operations are:

Fourier Transform

Description: An image is transformed to frequency domain by a discrete Fourier transform operation.

$$O(u,v) = \mathcal{F}(I(x,y))$$

where $\mathcal{F}$ is the Fourier transform.

Applications: Frequency filtering, removing periodic noise patterns and energy compacting.

Category: single-image.

Discrete Cosine Transform (DCT) Coding

Description: Pixel blocks (8x8 pixels) are discrete cosine transformed and then the frequency components are quantized.

$$O(u,v) = \mathcal{D}(I(x,y))$$

where $\mathcal{D}$ is the discrete cosine transform.

Applications: Lossy image compression.

Category: single-image.

Pattern Matching

Description: A block of image is compared to a set of blocks in terms of Euclidean distance to determine the best match between the blocks.

Applications: Documentation analysis, object recognition, vector quantization, motion estimation and fractal image compression.

Category: dual-image.

Image operations

The input for these operations consists of the intensity value of all the pixels (or the main part) of an image. Typical examples of these operations include:

Image Covariance and Correlation

Description: Image is modeled by random field representation.

$$Cov(u(m,n),u(m',n')) = E[(u(m,n)-\mu(m,n))(u(m',n')-\mu(m',n'))],\ \mu(m,n) = E[u(m,n)]$$

Applications: Image modeling and template matching.

Category: single-image.

Histogram calculation

Description: The relative frequency of each gray level in the image is calculated. The graph of the frequency as a function of gray levels is called the histogram of image.

$p_f(z)$ = *number of pixels with gray level equal to z.*

Applications: Image segmentation, measurement of textual properties and image comparison.

Category: single-image.

Huffman Coding

Description: Pixel values are replaced with variable-length codes based on their frequencies of occurrence in the image.

Applications: image compression.

Category: single image.

Mean square error / SNR computation.

Description: An image is compared to a reference image with these quantitative criteria.

$$MSE = {}^{1}\!/_{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} |u(m,n) - u'(m,n)|^2 , \; SNR = 10 \log_{10} {}^{\sigma^2}\!/_{MSE}$$

Applications: image comparison.

Category: Dual image.

2.4 Fractal Processing

The principal task in all of four different categories of image operations is to exploit the high correlation present in the visual data. Two new mathematical entities, namely **Fractals**[41] and **Wavelet Transforms**[96], have been recently introduced to exploit the correlation and self-similarities within an image or a sequence of images. Wavelet transformation belongs to the category of transform coding techniques, which attempt to exploit the correlation in an alternate domain rather than spatial domain. On the other

hand, Fractal processing extracts existing self-similarity and self-affine content within the image.

A majority of the processing operations listed above is accommodated in Fractal Block Processing (FBP)[38]. FBP is a computationally intensive procedure and involves operations such as, summation/accumulation, image addition/subtraction, translation, stretching, shifting, scaling, rotation and pattern matching. We have therefore chosen FBP as the candidate algorithm for the design of the generic video processing element which is detailed in chapter 6 of this thesis. The basic operations in FBP which are affine transformations are discussed in chapter 5.

2.5 MPEG4 Multimedia Standard

An emerging standard that is expected to become popular in visual domain processing (as well as other domains such as Audio) is MPEG4[16], [17]. MPEG4 is the third standard in a series developed by the Motion Picture Expert Group. The first two standards MPEG1[8], [14] and MPEG2[15] address the coding and compression of frame based video sequences and audio. MPEG1 was primarily used for Video-CD's with a resolution of 352x240. MPEG2 operated on a higher resolution (704x480) and has added support for interlaced video. Later, higher levels of resolution were specified so that MPEG2 could support HDTV. MPEG2 now supports all resolutions and frame rates defined by ATSC for digital television. MPEG4 was finalized in October 1998 as an ISO/IEC 14496 standard. MPEG4 differs from the previous standards in a number of ways. The new standard allows interactivity, high compression and accessibility to the video content. Video information in MPEG4 is no more specified as compressed frames but as

compressed Video Objects (VO). In this section, by introducing this standard, we show the validity of our categorization introduced in this chapter. We note that the operations involved in affine transformations, motion estimation and wavelet transforms will form important components in MPEG4. MPEG4 achieves a high performance by providing standardized ways to:

- represent units of aural, visual or audiovisual content, called "media objects". These media objects can be of natural or synthetic origin; this means they could be recorded with a camera or microphone, or generated with a computer as shown in Figure 9.
- describe the composition of these objects to create compound media objects that form audiovisual scenes;
- multiplex and synchronize the data associated with media objects, so that they can be transported over network channels.
- interact with the audiovisual scene generated at the receiver's end.

In addition,

(1) MPEG4 uses object based coding as opposed to frame and channel based coding of previous standards. MPEG4 also defines how interactivity between user and objects can be employed, in contrast to previous standards which allowed very limited interactivity. Objects in MPEG4 are very important because they allow content based interactivity. Objects are coded independently but grouped together to form a scene. Interactivity is enabled by the representation of a scene as a collection of objects or the composition. When the viewer selects or points to an object, actions that are

predefined for the object can occur. An object in MPEG4 is a component of a scene or the final audio-video presentation. Objects can be simple or compound.

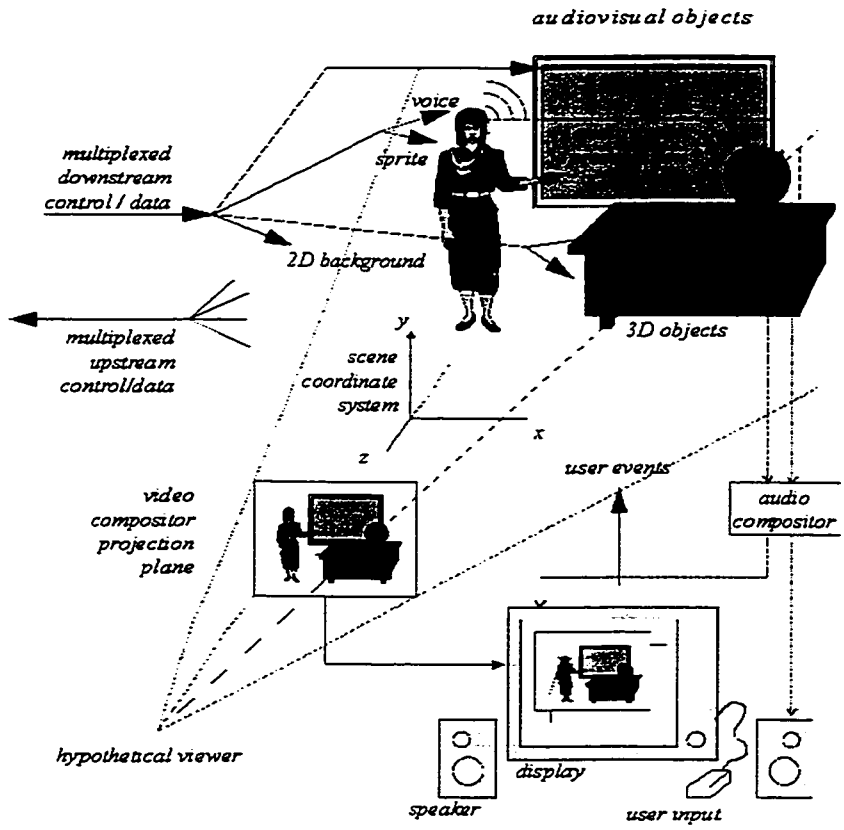


Figure 9 - An example of an MPEG-4 Scene

(2) MPEG4 allows the coding of objects as arbitrary shaped images or rectangular images. Previous standards can code only rectangular area. In MPEG4, arbitrary shapes are generated by coding a rectangular area and then adding a shape layer or mask which defines the exact shape of the video object.

(3) MPEG4 provides different coding methods (also called toolboxes) for coding different types of material. Computer generated or synthetic material can be coded using methods that are more appropriate to that format. MPEG1 and MPEG2 only addressed coding of natural material such as video or film. Normal video objects are usually coded using DCT based compression methods in MPEG4 similar to those used in MPEG1 and MPEG2. The DCT based coding methods are optimized for natural images that contain many shades of colors and smooth variations that normally occur in the real world. Computer generated images can have very few colors and many sharp transitions. DCT based coding methods do not compress these types of images. MPEG4 provides alternate methods of coding images that are computer generated. These objects are called synthetic objects to differentiate them from natural video and audio objects. Since computer generated objects can be created from sending commands to a rendering engine, one of the most efficient methods of compressing computer generated objects is to compress the commands to the rendering engine such as a text imager, 2D and 3D graphics rendering engine or sound generators.

(4) Because of the independent coding of objects in MPEG4, a means to combine objects in a scene is required. This is called composition and applies to both audio and video objects. Composition is the layering of objects to produce the final displayed image. Since objects can overlap depending on their size and position, it is required to determine which object is visible at any point in the displayed image. Further more, if a gray scale mask is used, the object being composited allows some of the underlying

objects to be visible as well. MPEG4 allows 3D **affine transformations** on each object before it is composited. This means that each object may be translated, scaled and rotated in 3D space before composition. The affine transformation parameters for each object can be modified during the presentation. Composition is defined by a scene description object in the MPEG4 bitstream. The scene is expressed as a hierarchy of nodes, which represent objects. The scene specifies audio-video composition as well as relationships between objects and the actions that can occur for each object.

(5) MPEG4 is designed to be an evolving standard. As new methods of coding are developed, they can be integrated in existing MPEG4 decoders so that they can decode new material, coded using the new algorithms.

(6) MPEG4 broadens applications from mainly two-way videophone appliances to a number of conceivable video communication or video entertainment devices.

(7) Since MPEG4 specifies the coding of natural and synthetic audio-video sources as independent objects, additional objects can be added to the mainstream without requiring the decoding and re-encoding. All the objects are combined in the decoder to form the final audio and video presentation.

We note that the novelty of MPEG4 resides in object based techniques. Affine transformations are among appropriate toolkits to achieve object based processing. Fractal processing is also an appropriate candidate for coding synthetic objects in a scene.

Wavelet coding is used to code texture information. In the next section, we present visual coding techniques, which are introduced in MPEG4 standard.

- Coding of Visual Objects

Visual objects can be either of natural or of synthetic origin. Different coding techniques are employed for different visual objects. In this section, we show that our proposed candidate algorithm encompasses a majority of coding techniques in MPEG4 standard.

2.5.1 Natural Textures, Images and Video

The tools for representing natural video in the MPEG4 visual standard aim at providing standardized core technologies allowing efficient storage, transmission and manipulation of textures, images and video data for multimedia environments.

The visual part of the MPEG4 standard will provide a toolbox containing tools and algorithms bringing solutions to natural visual objects. It will give an efficient representation of visual objects of arbitrary shape, with the goal to support so-called content-based functionalities. Next to this, it will support most functionalities already provided by MPEG-1 and MPEG-2, including the provision to efficiently compress standard rectangular sized image sequences at varying levels of input formats, frame rates, pixel depth, bit-rates, and various levels of spatial, temporal and quality scalability.

- Support for Conventional and Content-Based Functionalities

The MPEG-4 Video standard will support the decoding of conventional rectangular images and video as well as the decoding of images and video of arbitrary shapes.

The coding of conventional images and video is achieved similar to MPEG-1/2 coding and involves motion prediction/compensation followed by DCT based texture coding. We recall from section 2.4 that fractal/affine processing employs a super set of these operations.

Global motion compensation is based on the transmission of static "sprite" which is a possibly large still image describing panoramic background and motion vectors. For each consecutive image in a sequence, only 8 global motion parameters describing camera motion are coded to reconstruct the object. These parameters represent the appropriate affine transform of the sprite transmitted in the first frame.

- Coding of Textures and Still Images

Efficient Coding of visual textures and still images is supported by the visual texture mode of the MPEG4. This mode is based on wavelet transform that provides very high coding efficiency over a very wide range of bitrates. Together with high compression efficiency, it also provides spatial and quality scalabilities (up to 11 levels of spatial scalability and continuous quality scalability) and also arbitrary-shaped object coding. The wavelet formulation provides a scalable bitstream coding in the form of an image resolution pyramid for progressive transmission and temporal enhancement of still images. The coded bitstream is also intended for downloading the image resolution hierarchy into the terminal to be formatted as 'bitmap texture' as used in 3D rendering systems. This technology provides resolution scalability to deal with a wide range of viewing conditions more typical of interactive applications and the mapping of imagery into 2D and 3D virtual worlds.

- Scalable Coding of Video Objects

MPEG4 supports the coding of images and video objects with spatial and temporal scalability, both with conventional rectangular as well as with arbitrary shape. Scalability refers to the ability to only decode a part of a bitstream and reconstruct images or image sequences with:

reduced decoder complexity and thus reduced quality;

reduced spatial resolution;

reduced temporal resolution;

equal temporal and spatial resolution but with reduced quality.

This functionality is desired for progressive coding of images and video over heterogeneous networks, as well as for applications where the receiver is not willing or capable of displaying the full resolution or full quality images or video sequences.

2.5.2 Synthetic Objects

Synthetic objects form a subset of the larger class of computer graphics such as:

- a synthetic description of human face and body

⇒ The shape, texture and expressions of the face are generally controlled by the bitstream containing instances of Facial Definition Parameter (FDP) sets and/or Facial Animation Parameter (FAP) sets. Upon construction, the Face object contains a generic face with a neutral expression. This face can already be rendered.

- animation streams of the face and body

⇒ The rendered face is capable of receiving the animation parameters from the bitstream, which will produce animation of the face including expressions, speech, etc. Body Animation is being designed into the MPEG4 fabric to work in a thoroughly integrated manner with face/head animation.

- static and dynamic mesh coding with texture mapping

⇒ A 2D mesh is a partition of a 2D planar region into polygonal patches. The vertices of the polygonal patches are referred to as the node points of the mesh. MPEG4 considers only triangular meshes where the patches are triangles. A 2D dynamic mesh refers to 2D mesh geometry and motion information of all mesh node points within a temporal segment of interest. In 2D mesh based texture mapping, triangular patches in the current frame are deformed by the movements of the node points into triangular patches in the reference frame, and the texture inside each patch in the reference frame is warped onto the current frame using a parametric mapping, defined as a function of the node point motion vectors. For triangular meshes, affine mapping is a common choice. Its linear form implies texture mapping with a low computational complexity. Affine mappings can model translation, rotation, scaling, reflection and shear, and preserve straight lines. The degrees of freedom given by the three motion vectors of the vertices of a triangle match with the six parameters of affine mapping. This implies that the original 2D motion field can be compactly represented by the motion of node points,

from which a continuous, piece-wise affine motion field can be reconstructed. At the same time, the mesh structure constrains movements of adjacent image patches. Therefore, meshes are well-suited to represent mildly deformable but spatially continuous motion fields. The 2D object-based mesh representation is able to model the shape (polygonal approximation of the object contour) and motion of a VOP in a unified framework, which is also extensible to 3D object modeling when data to construct such models is available. 2D mesh modeling may be used for compression if one chooses to transmit texture maps only at selected key frames and animate these texture maps (without sending any prediction error image) for the intermediate frames. This is also known as self-transfiguration of selected key frames using 2D mesh information.

- Texture Coding for View Dependent applications

⇒ The view-dependent scalability enables streaming texture maps, which are used in realistic virtual environments. It takes into account the viewing position in the 3D virtual world in order to transmit only the most visible information. Only a fraction of the information is then sent, depending on object geometry and viewpoint displacement. This fraction is computed both at the encoder and at the decoder. This scalability can be applied both with Wavelet and DCT based encoders.

2.6 Summary

We have summarized various categories of operations in image and video processing followed by the derivation of generic operations for visual processing. Visual media processing involves operations to enhance, restore, compress and analyze images and video sequences. Image enhancement employs local correlation between adjacent pixels to enhance the image quality. In image restoration, it is assumed that the degraded image is a convolved version of the original image by a degradation function plus additive noise. Image compression techniques decrease the number of bits, which represent the image. Semantic correlation of pixels is used for image and video understanding in image/video analysis. We then present all operations in image/video processing. Our goal is to derive the fundamental operations and also a candidate algorithm to represent the majority of the operations in visual domain. Operations in visual processing are classified as follows:

- Point operations
- Local operations
- Line operations
- Geometric operations
- Block operations
- Image operations

The principal task in all of different categories of image operations is to exploit the high correlations present in the visual data using various operations. Fractal processing has been recently introduced to exploit the correlation and self-similarities within an image or a sequence of images. It is clearly shown in chapter 6 that fractal processing encompasses

a majority of visual operations. FBP (Fractal Block Processing) is a computationally intensive procedure and involves operations such as, summation/accumulation, image addition/subtraction, translation, stretching, shifting, scaling, rotation and pattern matching. Hence, we have chosen FBP as the candidate algorithm for the design of the generic video processing element which is detailed in chapter 6 of this thesis. Finally we have introduced novel techniques in the MPEG4 standard to show the validity of our proposed categorization and also to demonstrate the implementations of affine transformations. We now present enabling technologies for realization of Fractal Engine, in VLSI (Very Large Scale Integrated Circuits).

Ahma 1

3 Review of VLSI Technology

Multimedia hardware architectures are increasingly emerging due to advanced VLSI technology. Today's multimedia architectures are able to handle most of the required processing tasks for all of the media including image and video. VLSI technology[1] has grown exponentially in the last two decades. Powerful and integrated multimedia system implementations are now feasible due to recent advances in the VLSI area. The design of VLSI architectures for video processing is faced with a number of key choices. These include Integration (single chip VLSI, LSI, *etc.*), Fabrication Process (full custom, semi custom, *etc.*) and Design Tools (schematic capture, hardware description languages, *etc.*). We note that an efficient hardware design requires careful investigation of the state-of-the-art technology and choosing the tools that best suit the requirements of multimedia implementations.

3.1 Integration

VLSI (Very Large Scale Integration Circuit) is the technology of integrating million transistors onto a single device. The systems that required hundreds of discrete ICs in the past can now be designed into an IC that is about ¼ inch square. We note that it is not only the count of gates that determines the cost, but the number and types of ICs employed and the interconnection required to implement a digital circuit. Increased integration can offer reduced production costs as a result of high packing density, low system component cost and simplified assembly. However lower power dissipation, higher switching speeds and more system reliability are the other advantages.

Currently, chips with sub-micron features are quite common. For example, the 200-MHz Pentium Pro[58] and PowerPC 604e[59], [60], [61] have circuit features measuring only 0.35 micron across. The delivery of devices composed of 0.18-micron is now emerging. The number of transistors that designers can pack on a chip has increased at a rapid rate. For example, the logic density in the x86 processor family has increased 20 times in a span of 10 years as shown in the Figure 10.

The basis of these ever-higher logic densities is increasing levels of sophistication in photolithography[62]. Current lithographic processes employ a mercury light source whose 0.365-micron wavelength creates the 0.35-micron features. Successfully achieving the smaller 0.25-micron feature size requires the utilization of a krypton-fluoride ultraviolet laser that has a 0.248-micron wavelength. Smaller features are handled by the use of argon-fluoride lasers with a 0.193-micron wavelength. However, achieving 0.1-micron feature size requires optical trickery, which involves masks that phase-shift the light to improve the resolution. Building even-smaller chip features requires using light sources with even shorter wavelengths. In doing so, chip designers have traversed the electromagnetic spectrum from visible light, to ultraviolet light, and finally into X-ray territory.

Since multimedia processors require large number of devices to be packed onto a single chip, this high integration technology is crucial to support the development of chip-sets dedicated to this type of applications.

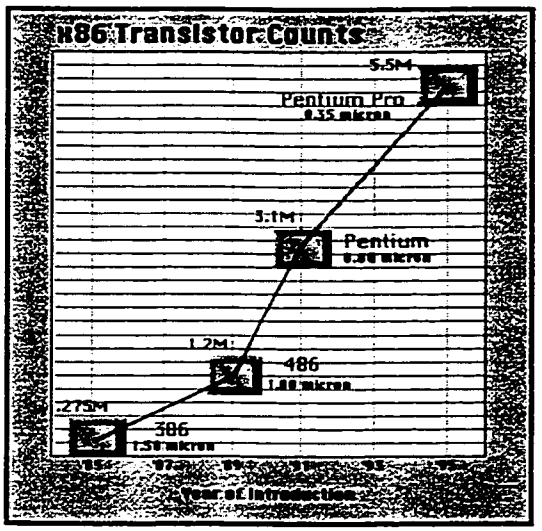


Figure 10- IC transistor counts.

3.2 Fabrication Process

Designed chip-sets can be fabricated using either a full custom or semi-custom design techniques[63]. The different choices include, full custom systems, cell-based systems, gate arrays and field programmable logic devices. The last three options are usually considered as semi-custom techniques and are distinguished by the name ASIC (Application Specific Integration Circuit). ASICs combined with new design tools (will be discussed in 3.3), have transformed the VLSI technology and made it possible for large numbers of designers to develop integrated circuits tailored to their specific application. Multimedia products are therefore made feasible due to this enabling technology. In this chapter, we discuss different options in hardware design.

3.2.1 Full Custom Design

Full custom design involves hand crafting of the chips at the silicon level and therefore demands a considerable amount of skill and experience on the part of the designer. Every individual transistor and connecting track has to be drawn in terms of basic geometric shapes (polygons) corresponding to features that will eventually be reproduced on the various mask levels for the silicon fabrication process. A typical process may require ten or more such masks to be produced. Drawing of the polygons is usually achieved using a graphics editor on a computer-aided engineering (CAE) workstation and is inevitably time consuming and error prone. The designer must observe a set of geometric design rules for the particular process that he/she is planning to use. At some stages it is necessary to verify that the layout that has been drawn conforms to these rules. When the layout of a cell is complete it is simulated at the transistor level. This process will include computation of track capacitance that is extracted from the physical description to yield accurate performance estimations. These steps are then iterated until satisfactory performance is achieved. Full custom design offers by far the greatest degree of flexibility of any of the techniques available. It gives the designer total freedom to decide what to integrate onto the chip (e.g. mixed digital and analog, power devices, special-purpose devices with integrated sensors). However, the time and effort involved can amount to many man-years and is justifiable only if production volumes exceed 100,000 units. Pentium MMX with extended multimedia instruction set is an example of this technology. We note that, the cost and the timing of this approach is not justifiable for our research.

3.2.2 Gate Array and Cell-Based Design

Gate array design offers the advantages of a custom approach but yet removes the need for labor-intensive transistor-level considerations, principally performance verification and physical layout, from the customer. To achieve this the silicon vendor carefully predetermines and characterizes a simple logic cell, typically having the potential to realize a few basic gates, and then repeatedly locates instances in a regular matrix covering most of the chip area. The gate array wafers are then fabricated as far as the interconnecting layer, typically representing 90% of the whole process; it is then left to the discretion of the user to determine a suitable pattern for a specific application. A number of architectural forms are available, being characterized by the pattern of cell layout and the amount of silicon explicitly devoted to interconnection paths. The gate array market is dominated by CMOS devices, which typically offer a few thousand gates with toggle rates up to about 350 MHz. Recent innovative families offer as many as 5,00,000 equivalent gates together with approximately 0.25 ns delay, and consequently the gate array technique now offers a high degree of versatility. To summarize, gate arrays achieve the objective of reducing design time compared to full custom devices, and require only a reduced customized mask set. Consequently they are appropriate for relatively small production volumes, typically a few thousand; in particular, prototyping using this medium is often attractive. Turnaround time for designs is typically a few months and a similar period is required if corrections are necessary. Consequently the importance of 'first-time correct' design is paramount. If sufficient turnover is anticipated, then time delays from completing a design to receiving a chip are possibly the major drawbacks associated with gate array design.

Cell-based IC design can be viewed as an attempt to obtain the best of both worlds (full custom and gate arrays). It offers the ease of design of gate arrays while retaining some of the density and performance advantages of full custom design. As with the gate array, the primary objective is to eliminate the need for the engineer to hand craft circuitry into silicon at the individual component or transistor level. This is achieved by making available to the chip designer a range of predefined and pre-characterized functional cells (collectively referred to as a cell library) which can be used as building blocks to construct any desired circuit. Cells can be drawn from the library as required and placed virtually anywhere on the silicon. The ability to optimize the cells represents one of the major advantages that cell-based systems have over gate arrays. We recall that the components in gate array cells are fixed in size and position by the manufacturer and consequently there is little or no scope for optimizing the manner in which these components can be connected together to realize a particular function. It is invariably the case that a given function implemented as a gate array cell will occupy a larger silicon area and have inferior performance compared with a hand crafted cell in a cell library. In terms of turn-around time a cell-based chip will demand equivalent effort to that required in fabricating a full custom chip. Compare this with the situation for gate arrays where almost one customized mask will normally be required to commit the array to a specific task. We note that in our project, a standard cell based technology of BiCMOS .8 micron is our primary selection for VLSI implementations of dedicated units. Fortunately this technology is available through CMC for our Lab.

3.2.3 Field Programmable Gate Arrays (FPGA)

The major disadvantages of gate array and cell-based design are the time taken to design and fabricate such a chip and the necessity for first-time correct solutions to minimize delays and costs. An attractive alternative allows for customization to occur in the field when all masking stages are complete. Programmable logic devices (PLDs) offer such a facility. They belong to the family known as "field programmable semi-custom". They consist of programmable logic gates that are connected through electronic fuses (switches). Programming a field programmable device means blowing the fuses (turning on the switch) along the path that must be disconnected (connected). Like traditional gate arrays, FPGAs implement thousand of logic gates. Field programmability is obtained at a cost in logic density and performance. FPGA capacity trails mask programmed gate array capacity by a factor of 10 and its speed trails mask programmed gate arrays by a factor of three.

On the other hand, a user can program an FPGA design in a few seconds or minutes, rather than the weeks or months required for the production of mask-programmed parts. Hence, FPGA design is a low risk design which makes FPGAs useful for rapid product development and prototyping. In addition, FPGAs can be fully tested after programming and hence user's designs do not require test program generation, automatic test pattern generation, and design for testability. Most FPGAs are now re-programmable and in the case of a requirement for modifications, they can be re-programmed within a few seconds.

Many kinds of programmable logic products are referred to as FPGAs. Here, we use a broad definition of the term, including not only devices with internal structure similar to gate arrays, but also devices with internal structure similar to a collection of PLDs. The term FPGA is often reserved for the former category, the latter are also called complex PLDs (CPLDs) or programmable multilevel devices (PMDs). Three programming technologies are commonly used for FPGAs. Each has associated area and performance costs, and the device architectures reflect those costs. Thus, we can categorize FPGAs as follows:

- ***Complex PLD (CPLD)***

In a CPLD architecture, the user creates logic and interconnections by programming EPROM (or EEPROM) transistors to form wide fan-in gates. A CPLD consists of a few function blocks, each similar to a simple two-level PLD. Each function block contains a PLD AND-array that feeds its macro-cells. The AND-array consists of a number of product terms. The user programs the AND-array by turning on EPROM transistors that allow selected inputs to be included in a product term. A macro-cell includes an OR gate to complete the two-level AND-OR logic and may also include registers and an I/O pad.

- ***SRAM FPGAs***

In an SRAM-programmed FPGA, programming of the device is stored in static memory cells. In SRAM FPGA, logic functions are implemented as lookup tables made from the memory cells, with function inputs controlling the address lines. Each lookup table of $2n$ memory cells implements any function of n inputs. One or more lookup tables, combined with flip-flops, form a configurable logic block (CLB). The CLBs are arranged in a two-

dimensional array with interconnect segments in channels similar to gate array architecture. SRAM FPGAs are inherently reprogrammable and can be updated in the system, providing designers with new design options and capabilities, such as logic updates that do not require hardware modification and time-shared virtual logic. Xilinx FPGAs are typical example of an SRAM FPGA.

- ***Antifuse FPGAs***

An antifuse is a two-terminal device that, when exposed to a high voltage, forms a permanent short circuit between the nodes on either side. Individual antifuses are small, so an antifuse-based architecture can have hundreds of thousands or millions of antifuses. To simplify the architecture and programming , antifuse FPGAs usually consist of rows of configurable logic elements with interconnect channels between them, much like traditional gate arrays. Typical example of Anti-fuse FPGA is Actel FPGAs.

3.2.4 Selected device

In our design process, we employ FPGAs in the implementation of control unit of the device which will bring flexibility and programmability to Fractal Engine. The selected target architecture is Altera / Xilinx SRAM FPGAs.

3.3 Design Tools

One of the important enabling technologies for successful ASIC development is having the proper design tools and a methodology that minimizes design errors at any level. In this section different design tools are studied with an emphasis on logic synthesis. Logic

synthesis is the design tool to analyze, verify, simulate and synthesize logic designs from a behavioral description to silicon implementation.

3.3.1 VHDL Synthesis

Hardware description languages[64] have shown to be essential parts in Logic synthesis. The rapid advances in integrated circuit technology over the past fifteen years have driven the need for more capable design tools, and as those tools have developed, they in turn have made it possible to design larger and more complex ICs. In the 1980s, a number of people within the electronics design community realized that conventional design tools and methods would be inadequate to handle the growing complexity and size of electronics systems. Two of the major advances to overcome that problem are the development of Hardware Description Languages (HDLs), and their use with powerful logic synthesis systems.

Logic synthesis is a process that is primarily intended to be used in the design of digital ICs, and, in particular, ASIC devices such as gate arrays. Logic synthesis or design automation is the automatic synthesis of a physical design from some higher-level behavioral specification which is much faster than manual design. It reduces the design cycle considerably, and allows the designer to experiment with various designs to obtain the optimal size/speed trade-off for a given application. Furthermore, as long as the original specification is verified and simulated, a synthesized circuit should not require either verification or simulation. High level behavioral specifications (the input to logic synthesis tools) are in general easier to write and to understand (and modify), less error-prone, and faster to simulate. Hence, they considerably facilitate the design of complex

systems. Today, synthesis is a growing industry, and commercial implementations of synthesis systems are widely used for production-level design of digital circuits.

Different levels of Logic synthesis are as follows:

- *High-level synthesis:* converts a high level, program-like specification of the behavior of a circuit into structural deign, in terms of an interconnected set of Register-Transfer level (RTL) components, such as ALUs (Arithmetic-Logic Unit), registers, and multiplexors.
- *Logic synthesis:* converts a structural design into optimized combinational (or sequential) logic, and maps that logic onto the library of available cells in a particular technology.
- *Layout synthesis:* converts an interconnected set of cells into the exact physical geometry (layout) of the design. It involves both the placement of the cells as well as their connection (routing).

An integrated synthesis system that covers all three synthesis levels is often referred to as a *silicon compiler*. Such a tool would allow the design of electronic circuits from a high-level, behavioral specification with little or no human intervention.

Hardware Description Languages (HDLs)[64] are used to describe the behavioral description of a circuit which is considered as the input to a high-level synthesis tool. Among all of HDLs, VHDL (VHSIC HDL)[32] has emerged as a standard for hardware specification and simulation. The development of VHDL was sponsored by the US government and the Air Force during the 1980s. In 1987, the VHDL language was

adopted by IEEE as a standard hardware description language and has since achieved wide spread industry acceptance. The VHDL language has powerful capabilities that have several possible functions depending on its application. The language can be used to describe and specify a variety of electronic systems, at levels of abstraction ranging from pure behavioral down to gate and switch level details. In addition to the description capability, systems modeled in VHDL can also be simulated at any of the levels in order to verify their functional operation and performance parameters. A number of very capable commercial VHDL simulators are available in the CAE marketplace. Finally, the VHDL description of a desired logic function can also be used to drive the logic synthesis process, with the constraint that the VHDL code be part of a fairly flexible but non-standardized subset of the language.

3.3.2 IC Design Methodology

Two conflicting forces drive the IC design process: circuit quality and time to market. We recall that semiconductor technology is undergoing exponential improvements, hence, rather than a single stable IC design methodology we see rapidly changing paradigm shifts as shown in the Figure 11.

- Transistor-Level Layout:

The premier IC design methods were focused on transistor level design coupled to layout. In this approach transistor level layout entry and transistor level simulation are employed.

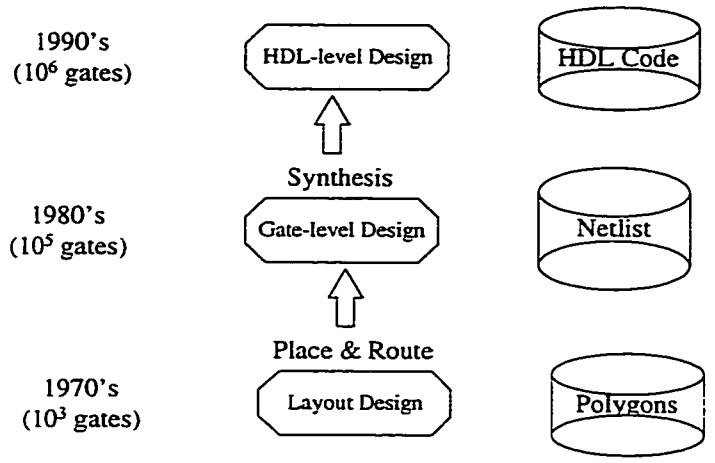


Figure 11 - Evolution of IC Design Methodology

- Gate-Level Entry:

Transistor level design is a time consuming process. The introduction of Gate Arrays (GAs), standard cells and Field Programmable Gate Arrays (FPGAs) brought comparable benefits to the IC designers. These IC technologies are supported by automated place-and-route systems. These systems take a net-list of cells from the library as input and automatically place and route them in rows and columns.

The utilization of standard cells, GAs and FPGAs in this approach raised the level of abstraction from the transistor level to the gate level. The primary design entry method is gate-level schematic entry by means of a schematic editor. Other key tools in this methodology are: gate-level simulation, automatic place-and-route tools and layout editors.

- Synthesis Based Design

We recall that the two important inventions of HDLs and logic synthesis systems accomplished this approach of synthesis based design. We now present different steps in synthesis based design process.

1. Behavioral Modeling:

For complex ICs, such as a high-performance microprocessor, a behavioral model of the IC is first developed. Behavioral modeling proposes modeling the functionally correctly, but without considering exact clock-cycle by clock-cycle behavior. This behavioral model can be expressed in a hardware description language such as VHDL.

2. Simulation and Testing:

The ability to fully test a behavioral model of a design is achieved by VHDL simulators. The code for the VHDL descriptions and test patterns will normally be typed as ASCII text files which are the input source for the VHDL simulation tools. The reason that this can be accomplished so quickly is that the synthesizable VHDL code is written at a fairly high level compared with the gate by gate details required on a schematic, and this itself takes much less time. Hence, with VHDL, the simulation begins immediately which enables to find design problems in early stages.

3. Logic Synthesis:

Although synthesis is a fairly automated process, additional details must be provided to the tools. First is the decision of which ASIC vendor will be used. Hence, the vendor specific library of cells and parts are required in order to generate the gate level design. This library normally contains the details of individual gate delays and the rules for

computing loading delays due to inputs and estimated capacitance and wire length. The second input to the synthesizer is information that is used to constrain the design based on designer's requirements. This typically includes the clock rate and pulse width, assumptions about operating temperature, voltage, and process variations, output loading, and limits on permissible propagation delays through critical paths. The outputs of the synthesis tools typically include a vendor specific net-list, reports on timing, gate count and area, critical paths and plotted schematic diagrams. Figure 12 shows the detailed design flow for two different methods.

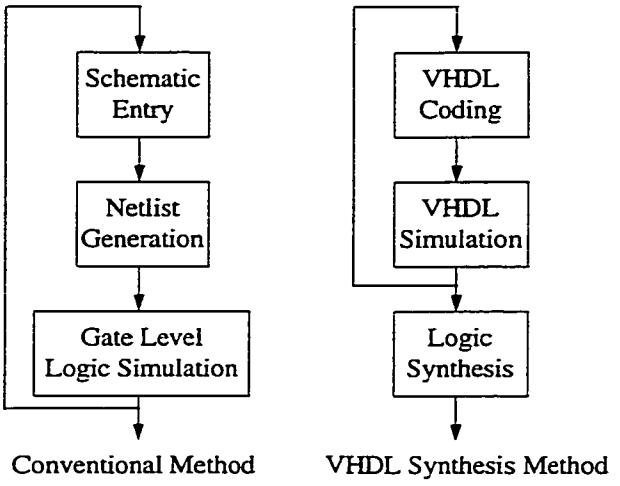


Figure 12- Comparison of Design Flows

In this thesis, our goal is to obtain a VLSI system, which meets the performance and specification requirements for real-time implementation of multimedia applications. To achieve this objective, we have chosen to implement all hardware designs in behavioral description in early stages using VHDL. Primary validation of functionality is assured by VHDL simulators. Logic synthesis is then applied and vendor specific net-list is generated (as shown in Figure 13). Real parasitic values, routing and propagation delays

are then back-annotated and final simulation assures the functionality of the design. The target VLSI technology is standard cell based for more critical modules and FPGA for the programmable control unit.

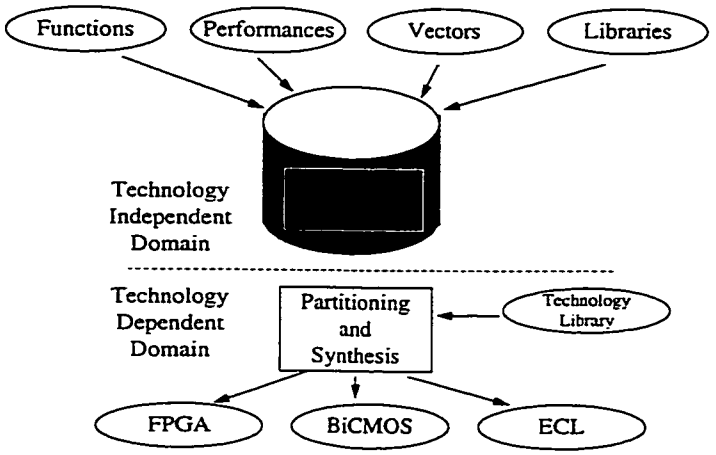


Figure 13 - VLSI Design Process

In the next chapter, architecture design trends are discussed in order to select the best strategy for Fractal Engine.

3.4 Summary

The design of VLSI architectures for video processing is faced with a number of key choices. These include Integration (single chip VLSI, LSI, *etc.*), Fabrication Process (full custom, semi custom, *etc.*) and Design Tools (schematic capture, hardware description languages, *etc.*). High-density VLSI chip-sets require new design automation systems. They can be fabricated using either a full custom or semi-custom design techniques. The different choices include, full custom systems, cell based systems, gate arrays and field programmable logic devices. In our research, we employ cell based system design

techniques for dedicated units and field programmable gate arrays for programmable units.

Logic synthesis using VHDL entry codes is our main design tool which minimizes design errors at any level. Logic synthesis is the design tool to analyze, verify, simulate and synthesize logic designs from a VHDL behavioral description to silicon implementation.

In addition to high density and fast VLSI systems, multimedia systems require new advanced techniques in parallel processing. We present different aspects of multimedia hardware architectures in the next chapter. Our goal is to find the best design scheme for Fractal Engine.

4 Design Trends in Multimedia Architectures

Multimedia system design presents challenges from the perspectives of both hardware and software. Each media in a multimedia environment requires different processes, techniques, algorithms and hardware implementations.

Multimedia applications require efficient VLSI implementations for various media processing algorithms. Emerging multimedia standards and algorithms will result in hardware systems of high complexity. In addition to recent advances in enabling VLSI technology for high density and fast circuit implementations (discussed in Chapter 3), special investigation of architectural approaches is also required. The important issues in multimedia hardware design are listed below:

- Parallelization and Granularity: MIMD, SIMD, coarse grain such as multiprocessor architectures and fine grain like superscalar and VLIW[87] architectures, etc.
- Processor (datapath) choices: DSP, RISC and CISC[65].
- Memory Interface Design: Support for EDO-DRAM, SDRAM, VRAM, RDRAM, etc.
- Flexibility: Dedicated or programmable.

In this chapter, we investigate different architectures and categorize them. We note that some categories are not restricted to multimedia processors. After the review of all architectures, we analyze advantages of each technique to be employed in Fractal Engine.

4.1 Flexibility

In general, there are two different approaches for multimedia architecture design (Figure 14) as of any core processor namely: Dedicated and Programmable. Combination of dedicated and programmable modules in a multimedia architecture offers a compromise between the two strategies as an adapted architecture for multimedia purposes.

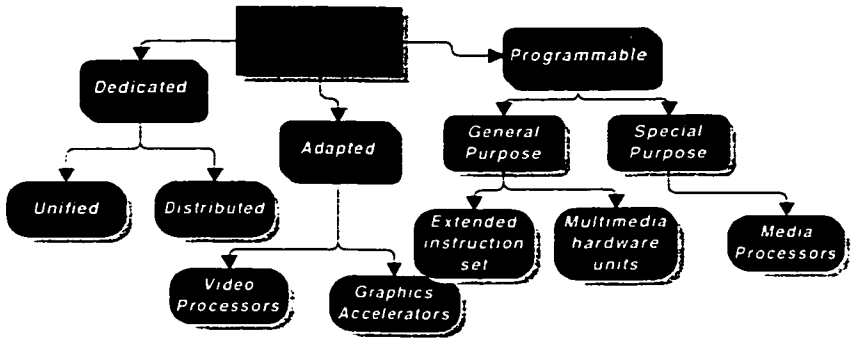


Figure 14- Multimedia Architecture Trends

A function specific (dedicated) implementation is a direct mapping of the multimedia processing tasks to hardware modules optimized to execute the specific functions. Matching of the individual hardware modules to the processing requirements results in area efficient implementations. Multimedia programmable processors consist of operational and memory modules, which enable the processing of different tasks under software control. Combination of dedicated and programmable modules in a multimedia architecture offers a compromise between the two strategies. As shown in Figure 14, the architectures range from dedicated and adapted modules to fully programmable media processors. A brief description of each category is presented in the following sections.

4.1.1 Dedicated Architectures

Based on available technologies, required computational achievement, production quantity and the target algorithm the use of dedicated implementations could become the best choice. For high volume consumer products, the optimization in silicon area and timing of the device, which is brought by dedicated architectures, decreases the production cost. Also, designing an specific function architecture for a well defined and established standard algorithm is the best alternative. Dedicated processors differ in terms of the ability of computations. They range from a small module for a specific small task such as a DCT chip to a complete MPEG-2 encoder, which are discussed in the following sub sections.

4.1.1.1 Distributed (Chip-Set) Implementation

In a chip-set, each major video processing module is configured as a separate chip such as a DCT chip, Huffman coder chip, motion estimator chip, etc. Each module is designed by a dedicated hardware architecture. In a distributed implementation, the designer is responsible for the interconnection of the chips. The advantage of this approach is the flexibility in selecting and connecting the different modules. The disadvantage is the increase in area and therefore the size of the system. A typical distributed implementation is shown in Figure 15. LSI Logic's L64735 DCT Processor Chip and L64765 Color and Raster/Block Converter Chip[89] are good examples of this approach.

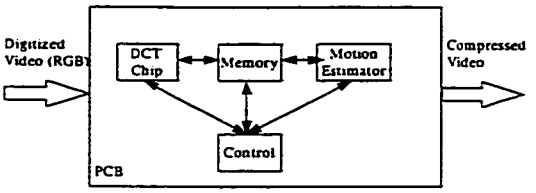


Figure 15- Distributed implementation example.

4.1.1.2 Unified Implementation

In this approach the whole system is designed in a single chip (or chip set) which results in a low power dissipation and reduced silicon area. The main disadvantage of this approach is the lack of the flexibility. Figure 16 shows a typical unified implementation. An example of this approach is the C-Cube CL451[90].

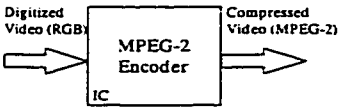


Figure 16- Unified implementation example.

4.1.2 Adapted Architectures

The idea of designing a more flexible architecture for multimedia applications is necessary because of the increasing number and variety of multimedia applications. Dedicated architectures fail to respond to any change in the implemented algorithm. Most dedicated architectures for multimedia processing applications achieve an increase in flexibility by an adaptation of a programmable architecture to the algorithmic requirements. Visual media being the most complex media in a multimedia environment, has been the main target of architecture adaptation. Graphics and video chips have been specifically designed the details of which are now discussed.

4.1.2.1 Graphics/3D Accelerators

The graphics accelerator chip (or chip set) is designed to perform computationally intensive tasks by providing hardware acceleration for the execution of low-level graphics operations. Hence, they often function as a coprocessor in workstations and personal computers. Newer chip sets often include hardware assistance for displaying 3-D data and video streams. A typical system with a graphics accelerator is shown in Figure 17. An example of this category is the ViRGE/VX by S3 [92].

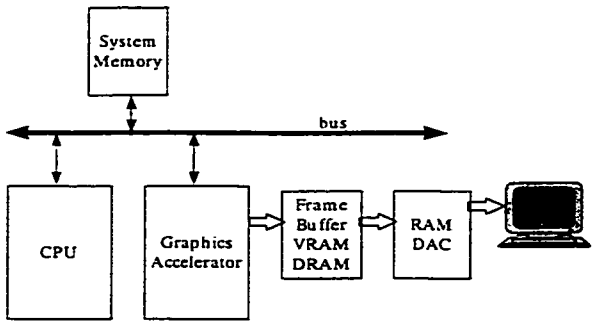


Figure 17- A typical graphics accelerator system

4.1.2.2 Video Processors

Video processing tasks, such as DCT, motion estimation and variable block coding, demand a high performance processor. Most processing units accomplish higher speeds with the aid of a video coprocessor, which is capable of execution of above-mentioned tasks. Recently, several video processors have appeared in the literature including the VCP by 8x8 [91]. Figure 18 illustrates the utilization of a video processor in a complete system.

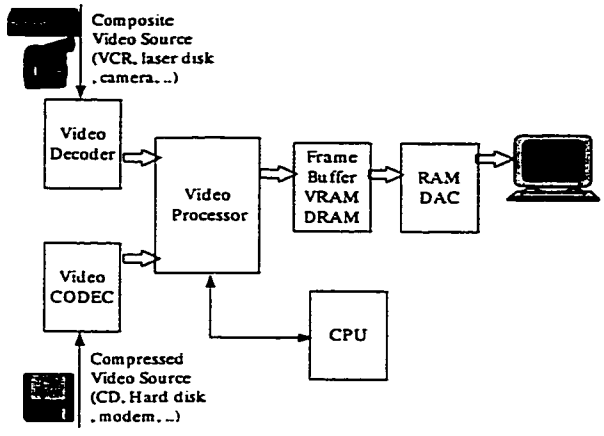


Figure 18- Video processor implementation

4.1.3 Programmable Architectures

In contrast to function specific approaches with limited flexibility, programmable architectures implement different tasks by software control. The main advantage of programmable architectures is the increased functionality. The design of a programmable multimedia processor could be based on the design of a general purpose architecture or performed independently for multimedia applications. In the former case, multimedia capability add-ons are realized either in extending the instruction set or adding multimedia hardware units. In the latter approach, a processor is specifically designed for multimedia purposes. These alternatives are discussed in the following sub-sections.

4.1.3.1 General Purpose Processor with Extended Multimedia Instruction Set

Adapted architectures like graphics and video cards in workstations and personal computers have the disadvantages of increased cost to the end user. General purpose processors including RISC and DSP have significant computing power but are not optimized for multimedia processing. Therefore, there is a strong desire among computer

manufacturers to enhance existing architectures so that multimedia processing (video and graphics processing) is integrated into the next generation processors just as 2D graphics processing has been integrated into today's architectures. Extended multimedia instruction set which is introduced by Intel in MMX™[27] processors is an example of this approach.

4.1.3.2 General Purpose Processor with Multimedia Hardware Units

The previous approach does not optimize the hardware for multimedia applications with highly intensive computations. By using the enabling VLSI technology the alternative solution is to add dedicated multimedia hardware units to the processor. The MediaGX processor [93] is an example of this approach. MediaGX not only executes x86 instructions using a Cyrix CPU core, it also acts as a virtual video card resulting in a highly integrated device with a lower cost and superior performance.

4.1.3.3 Media Processors

Media processors are a new category of logic devices defined as software-programmable processors that are dedicated to simultaneously accelerating several multimedia data types. Media processors meet three requirements:

1) software-enabled (not a multi-function fixed-function ASIC);

2) dedicated to multimedia (not multimedia extensions to a CPU, like MMX™);

3) capable of accelerating several multimedia functions simultaneously (not a DSP).

Recent advances in multimedia technologies such as DVD (MPEG-2 video, Dolby Surround AC-3™ audio), 3D graphics, home movie editing (MPEG encoding), and

video-phone, involve computationally intense operations and hence make it expensive and difficult to design a dedicated chip or add-in boards for every new technology. Hence, media processors are the target of new multimedia designers. A typical system implementation by a media processor is shown in Figure 19. There are several vendors now in the process of media processor design. TriMedia[29] by Philips is an example of a media processor.

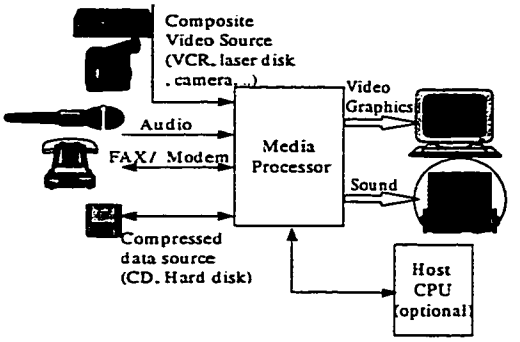


Figure 19- Typical media processor system

In the following sections, other options in the design of a multimedia hardware architectures are discussed. We note that these options mostly are applicable to programmable multimedia processors and are used widely in today's architectures.

4.2 Processor selection

Programmable architectures have several units in common. In general, every programmable architecture consists of data path, memory, input/output and control path. Data path is responsible for all the operations performed on data for the purpose of data input, manipulation, analysis, processing and output. Control path is generating all necessary signals to control the interaction between modules. There is always a contest

between the complexity of these two parts in the design of a processor. The larger data path leaves less space for control path and vise versa.

In this section, we investigate the possible options for the design of the processor[65] in a multimedia system as shown in Figure 20. The categorization scheme is based on the format of instruction-set, available registers and the structure of data path.

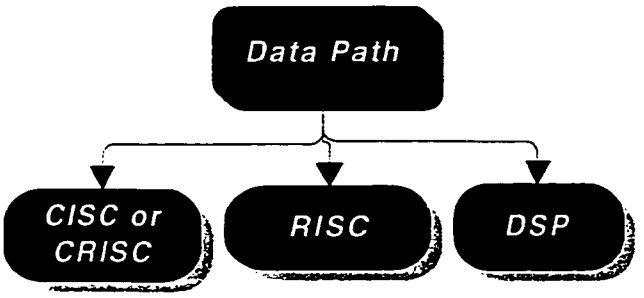


Figure 20- Data path selection

CISC (Complex Instruction Set Computing) microprocessors with a more complex instruction set provide more direct hardware support for a software developer than any other architecture. Instructions in a CISC processor are very powerful in terms of processing capability and support large numbers of registers and addressing modes. Control path in CISC processors is more complex in order to execute instructions that are more powerful. RISC (Reduced Instruction Set Computing) microprocessors offer faster execution of individual instructions by optimizing the processor for a smaller instruction set. DSP (Digital Signal Processing) microprocessors are optimized to perform digital processing operations such as filtering. Multiply and Accumulate (MAC) instruction occurs frequently in DSP algorithms and is performed in one cycle in DSP processors.

4.2.1 CISC/CRISC

In early stages of microprocessor design, memory sub-systems were far slower than the processor (this gap though narrower still continues today). In order to decrease the memory access by CPU, engineers designed complex instruction sets. Each instruction encapsulates several simple instructions, and hence the time spent retrieving the instruction from memory was reduced. Another design key for CISC processors is microprogramming. Microcode essentially acts as a translation layer between the instructions and the data path. In a microprogrammed system, the main processor has some built-in memory (typically ROM) which contains groups of microcode instructions which correspond to each instruction. When an instruction is retrieved by the processor, the processor executes the corresponding series of microcode instructions. Using microprogramming, designers are able to update a processor with new instruction sets without changing the hardware. Since the microcode memory can be much faster than main memory, an instruction set can be implemented in microcode without loss of speed over a purely hard-wired implementation.

The Characteristics of instruction sets in CISC processors include:

- Register to register, register to memory, and memory to register commands.
- Multiple addressing modes for memory, including specialized modes for indexing through arrays.
- Variable length instructions, where the length often varies according to the addressing mode.

- Instructions which require multiple clock cycles to execute.

Two key features of CISC hardware architectures are:

- Complex instruction-decoding logic (complex control path)
- A small number of general purpose registers.

4.2.2 RISC

Analysis of the instruction mix generated by CISC shows that about 80% of the instructions generated and executed uses only 20% of the instruction set. It is an obvious conclusion that if this 20% of instructions is speeded up, the performance benefits would be far greater. Further analysis shows that these instructions tend to perform the simpler operations and use only the simpler addressing modes. Essentially, all the effort invested in processor design to provide complex instructions and thereby reduce the compiler workload is being wasted. Hence, if only simpler instructions are required, the processor hardware required to implement them could be of reduced complexity. It therefore follows that it is possible to design a more powerful processor with fewer transistors and lower cost. This processor has a simpler instruction set and hence, executes its instructions in a single clock cycle and synthesizes complex operations from sequences of instructions. The main features of a RISC processor are as follows:

- All instructions will be executed in a single cycle.
- RISC processor must include pipelining techniques to segment instructions.
- Memory will only be accessed via load and store instructions.

- All execution units will be hardwired with no microcoding.
- On-chip instructions and data cache stores often used to decrease memory access.
- Operations are register based.

➢ ***CRISC***

The advent of new processors, which combines the advantages of both RISC and CISC architectures, has made distinction between CISC and RISC architectures no longer clear-cut. Now a processor capable of executing multiple instructions in a cycle contains a large instruction set of over 200 instructions and therefore cannot be considered as a RISC processor. Typical examples of this category are the PowerPC and Pentium processors.

4.2.3 DSP

DSP processors are optimized for digital processing operations which include multiply and accumulate (MAC) operations. MAC operation $r = b + a \cdot x$, requires multiple clock cycles in CISC and RISC processors, while in a DSP it is executed in one clock cycle. Some characteristics of a DSP processor include:

- Multiple data and instruction buses.
- Parallel execution of MAC operation.
- Limited number of instructions.
- Efficient loop control.

4.3 Granularity

The granularity of a multimedia system defines the size of the individual processing units by which tasks are executed. The granularity affects the number of processing units since, in any parallel architecture there is a tradeoff between the size and the number of processors.

Coarse grain systems are formed by a small number of large and complex processing units. In fine grain parallelism, there is large number of small processors. The intermediate possibilities between these two extremes can be referred to as medium-grain parallelism. Fine and medium grain parallelisms have the potential of being faster, but they need more powerful control units to divide small tasks between the processing units efficiently. Most of multimedia processors are categorized as fine/medium grain processors. Each task is executed in parallel at the instruction level among several processing units. We now present the scheme of parallelism employed in these machines and we will present data level parallelism in section 4.4.

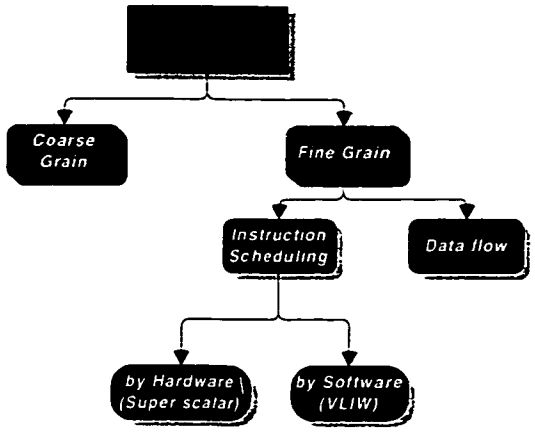


Figure 21 - Granularity issue in multimedia architectures

4.3.1 Instruction Scheduling – Super scalar

The objective of a super scalar[66]-[69] system is to execute more than one instruction in each clock cycle. The basic idea is to build a processor whose data path includes multiple functional units and a modified control path to divide each task among the functional units and keep them busy as much as possible. For example, a data path is formed by several adders and a couple of multipliers. Thus the processor is able to perform a number of additions and multiplications at the same time. To achieve this, the control unit should be able to analyze a sequence of instructions and decide when some of them can be executed in parallel. In a super scalar machine, the central processing unit (CPU) manages multiple instruction pipelines to execute several instructions concurrently during a clock cycle.

4.3.2 Instruction Scheduling – VLIW

VLIW (Very Long Instruction Word) processors [69]-[71] achieve instruction level parallelism through software control in contrast to super scalar architectures. A VLIW instruction is a long string of bits (few hundred to few thousand bits) that directly controls every individual processing element in the processor. Each bit could turn on or off a particular element of the data path. Parallel execution is arranged simply by setting the instruction bits that activate several functional units at the same time. The hardware does no instruction scheduling; all decisions about controlling the functional units must be made when the program is compiled. Hence, VLIW architectures are data path intensive and require low control complexity.

4.3.3 Data flow

Data flow architectures[72] achieve parallelism based on the concept of executing program instructions as soon as their operands are ready, instead of following the sequence dictated by instruction code. The data flow architectures could be massively parallel. Control functions are placed on the data side (data-driven). The architecture can eliminate the need for a processor clock and hence the processor has extremely low power consumption. The architecture itself has power management functions so that it operates only when data is present in the computational section.

4.4 Data distribution

We recall from Section 4.3 that in fine/medium grain systems, parallelism can be achieved by either task distribution (instruction level parallelism) or data distribution. In data distribution parallelism, the data is distributed among several processing units which perform operations in parallel over the different data segments. Processors are classified according to how they process the program instruction and data streams, namely (i) SISD – Single Instruction Single Data, (ii) MISD – Multiple Instruction Single Data, (iii) SIMD – Single Instruction Multiple Data, (iv) MIMD – Multiple Instruction Multiple Data[86]. It is clear that the last two classes employ data distribution for parallelism and hence, they are discussed in this section.

4.4.1 SIMD

In SIMD architecture, all processing units execute the same instruction in the same machine cycle over different data. They include vector/array processors, associative and

orthogonal processors. A control unit issues the execution command to all processing units and hence the control design is simple.

4.4.2 MIMD

A MIMD architecture typically achieves high utilization of all processing units. It needs separate control units and instruction memories per parallel unit. Compared to SIMD, the advantage of MIMD is greater flexibility and higher performance for complex algorithms with highly data dependent control flow. On the other hand, MIMD requires a significantly increased silicon area. Additionally, the access rate to the program memory is increased, since several controllers have to be provided within program data.

4.5 Memory Selection

Multimedia applications with large volumes of data require very large memory bandwidth. Hence, high density, fast and low power storage is an essential part of each multimedia system. Also, the clock speed in processing units has been increased and a fast memory is required to match the processing speed. In order to meet these requirements, several approaches have emerged recently which increase the performance of DRAM memories. These techniques include extended data out (EDO) DRAM, synchronous DRAM (SDRAM), Rambus (RDRAM) Dram and video (VRAM) DRAM.

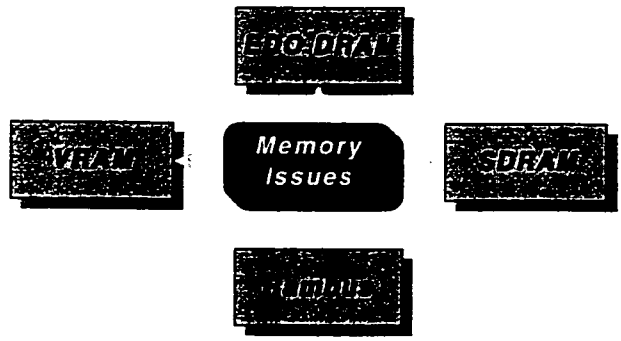


Figure 22 - Available DRAM options

4.5.1 EDO RAM

In EDO (Extended Data Out) memories, output data can be maintained until the next CAS (column address strobe) falling edge. This results in continuos memory accesses. DRAM has a two-stage pipeline, which lets the memory controller read data off the chip, while it is being reset for the next operation.

4.5.2 SDRAM

SDRAM (Synchronous DRAM) is another form of memory developed shortly after EDO. Performance improvement of SDRAM is achieved by introducing synchronous operation to DRAM. Because of being in sync with the processor, it eliminates timing delays and makes the memory retrieval process much more efficient.

4.5.3 Rambus DRAM (RDRAM)

RDRAM is an interface design in order to provide an optimized solution for data transfer between memory and processor. It adopts a 9-bit data bus, and there is no dedicated address bus. Instead, packets including both command and address are first sent to the chip via the Rambus channel. Following the request packets, an acknowledge packet and

a data packet are sent from the chip back to the controller. After initial latency, data is accessed at high speed.

4.5.4 VRAM

Graphics memory must work very quickly to update, or refresh, the screen (60-70 times a second) in order to prevent screen flicker. At the same time, graphics memory must respond very quickly to the CPU or graphics controller in order to change the image on screen. With ordinary DRAM, the CRT and CPU must compete for a single data port, causing a data traffic bottleneck.

VRAM (Video RAM) is a dual-ported memory that solves this problem by using two separate data ports. One port is called the serial access memory (SAM) dedicated to the CRT, for refreshing and updating the image on the screen. The second port which is the random-access port is dedicated for use by the CPU or graphics controller, for updating the image data stored in memory.

4.6 Multimedia Processors

In this section, we introduce example processors for multimedia applications. The objective is to show the validity of categorizations discussed in this chapter. These processors are designed for different target applications and one example is selected for each application. Examples include:

Application	Example:
1. Multimedia Video Processor	MVP by Texas
2. Generic Media Processor	Mpact 2 by Chromatic
3. Generic Media Processor	TriMedia by Philips
4. Embedded multimedia processor	V830R/AV by NEC
5. Dataflow Media Processor	DDMP by Sharp
6. General Purpose Processor	Pentium with MMX technology by Intel
7. Video codec for studio applications	VideoRISC by C-Cube
8. Audio/Video codec (MPEG-2)	L64002 by LSI-Logic
9. Graphic and video processor	ViP by IBM
10. Video conferencing solution codec	VCP by 8x8
11. Image compression and motion estimation	ICC and MEC by Array Microsystems

Parallel processing techniques with multiple processing elements and memory system, which typically communicate through an interconnection network are employed in these architectures.

4.6.1 TI MVP

In 1994, TI introduced the TMS320C80 single chip Multimedia Video Processor (MVP)[76]. MVP combines, on a single semiconductor chip, multiple fully programmable processors with multiple data streams connected to shared RAMs through a crossbar network. Each of the independent processors can execute many operations in parallel in every cycle. MVP has a scalable architecture with an overall performance of 2 MOPS (million operations per second). Figure 23 shows a block diagram of the major functional blocks of the MVP. The Master Processor (MP) is a RISC processor with an integral floating-point unit. MP is used primarily for host interface, sequential processing, and management of multiple concurrent tasks operating on the entire MVP.

The MVP's advanced DSPs have a unique parallel architecture optimized for image and video computing. These DSPs have many powerful features not found in conventional DSPs, such as:

- Long instruction words (64 bits): allowing up to 15 RISC-equivalent operations to be specified in a single instruction.
- Single-cycle parallel accesses to the on-chip memory: allowing two 32-bit data transfers per processor in every cycle, concurrent with data operations.
- Three-input 32-bit ALU, which can be optionally split into two 16-bit units or four 8-bit units.
- 16x16 multiplier, which can also be split into two 8x8 units.

- Dedicated adders for address generation, which can also be used for arithmetic operations.

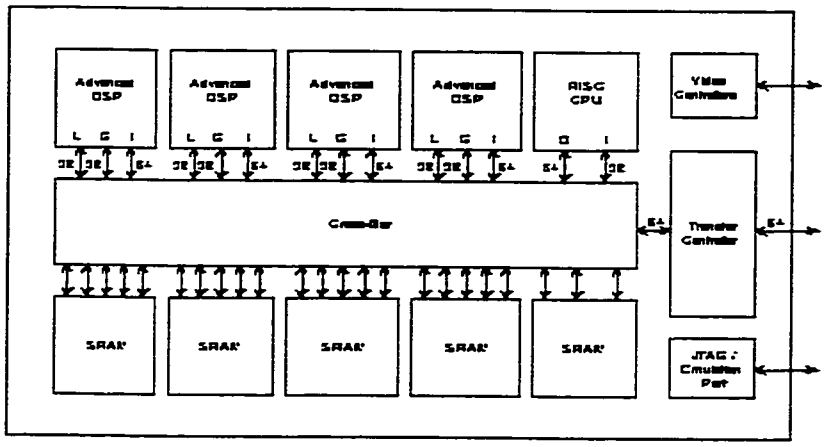


Figure 23- MVP Block Diagram

The MVP also includes 50 K-bytes of on-chip SRAM accessible in a single cycle. The memory is organized as 25 blocks of 2 K-byte modules and each module used as an instruction cache, data cache, data RAM, or parameter RAM. An instruction cache is assigned to the MP and each of the DSPs, while the data cache is available only to the MP. For the DSPs, the data RAM serves as the local storage area. While the cache memory is serviced automatically in hardware by the Transfer Controller (TC) for transfers to and from the external memory, the data RAM needs explicit management and requests to the TC by the processors in software. Each DSP is associated with 8K-bytes of on-chip RAM modules, although any processor can perform a single-cycle access to any data or parameter RAM module via the crossbar. The TC is an intelligent DMA controller, responsible for interfacing to the external memory system. It prioritizes different types of data transfer requests from the MP and the DSPs, and transfers the data

within or between the on-chip and external memories. It has numerous modes of transfer operations, such as multi-dimensional transfers, table-guided transfers, fill-with- value, and serial register transfers (SRT).

The processors and the memory modules are fully interconnected through the crossbar which can be switched at the instruction clock rate (20 ns). Inter-processor communication protocols such as message passing and pipelining can be easily implemented in software, since each memory access takes only one cycle. In the case of simultaneous access to the same location, the crossbar connections ensure that such contentions are resolved through a priority-based scheduling.

The MVP also integrates the Video Controller (VC) for the generation of video timing signals and VRAM memory transfer cycles, eliminating the need for external circuitry and thus reducing the board space and the number of chips needed in video systems.

4.6.2 Chromatic Research Mpact 2

The block diagram of Mpact 2[77] is shown in Figure 24.

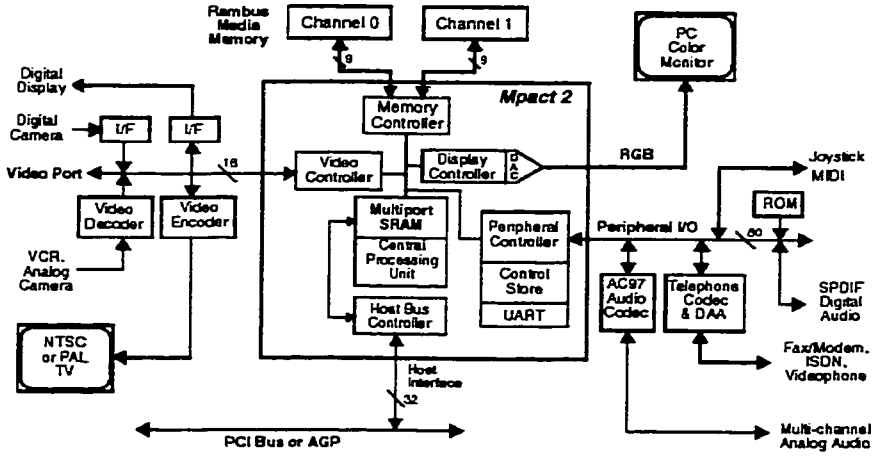


Figure 24 - Mpact 2 block diagram

Mpact 2 is a media processor designed for multimedia applications in PC. The Mpact 2 chip consist of a signal processor and five DMA bus controls. Data is transferred simultaneously between the memory and the bus system. It includes dual Rambus channels capable of a date transfer rate of 1.2 Gigabytes per second. The is a VLIW architecture with a SIMD control unit. Data paths are all 72 bits wide. There are on-chip caches for instruction and data. Data cache is a multiport memory with six read and six write ports. AGP and PCI interfaces are designed in this chip and are readily available.

4.6.3 Philips TriMedia TM-1000

TM-1000[29] is the first media processor from the family of TriMedia processors. The core processor inside TM-1000 is a high performance VLIW-based CPU core. The core incorporates 27 functional units. The selection of the functional units is based upon the application. Every VLIW instruction is formed by a maximum of five operations. The core has 128 general-purpose 32-bit registers. There are 15 read ports and five write ports in the register file.

TM-1000 processor consists of memory, video, audio modem and PCI interfaces which makes possible easy communication with multimedia devices as shown in Figure 25.

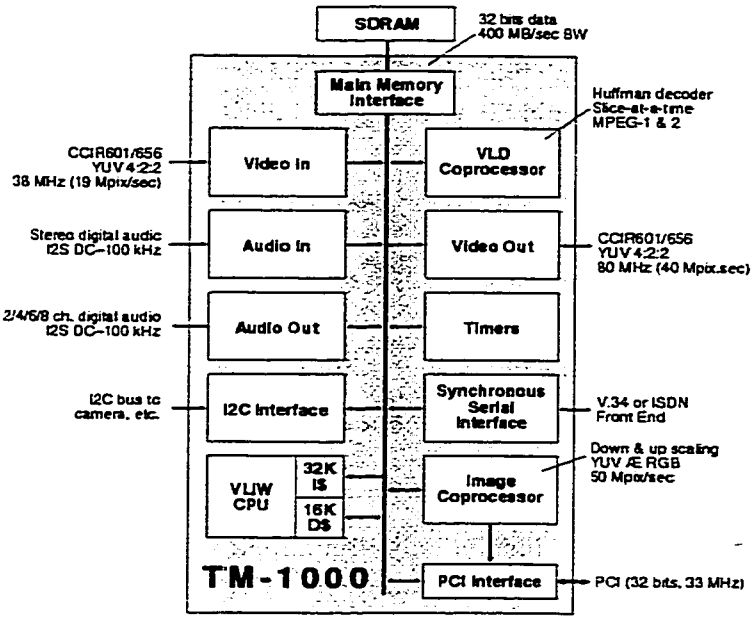


Figure 25 - Block diagram of TM-1000

4.6.4 V830R/AV by NEC

V830R/AV[78] is an embedded multimedia processor designed for low cost multimedia oriented applications. It is targeted to support real-time video signal processing of broadcast quality. Strong multimedia processing extensions are incorporated into V830 RISC engine, which is the base of V830R/AV processor. The core architecture supports 32-bit MAC operations. The processor is based on a two-way superscalar architecture. The two major execution units in the V830R CPU core namely, the 32-bit integer execution unit and a 64-bit multimedia extension unit, can work in parallel to improve the performance. This 64-bit multimedia coprocessor performs SIMD parallel operations on eight bytes, four half-words, or two words packed in thirty-two 64-bit coprocessor registers. The execution units are fully pipelined and have one clock throughput and fixed 4-clock latency. The key features of V830R include:

- Dual-issue superscalar
- Rambus interface ready
- 16K four way instruction and data cache
- video/audio, DMA, A/D multiplexed bus and ICE interfaces.

The V830R CPU core has a six-stage pipeline structure. The whole pipeline is divided into three pipelines: an Instruction pipeline (I-pipe), an integer pipeline (V-pipe) and a multimedia pipeline (M-pipe). The processor is capable of executing MPEG-2 decoding in the main profile at main level (MP@ML).

4.6.5 Sharp DDMP

DDMP[79] (**D**ata-**D**riven **M**edia **P**rocessor) is the first data flow processor designed for multimedia applications. This device uses high-speed parallel processing techniques to process massive amounts of multimedia information, including full-motion video, graphics, and audio.

The DDMP puts control functions on the data side (data-driven) and eliminates the need for a processor clock in contrast to conventional von-Neuman computers. The result is a media processor with extremely low power consumption in which the architecture itself has power management functions so that it operates only when data is present in the computational section. The DDMP media processor consists of a number of cores, controllers and I/O circuitry as shown in Figure 26.

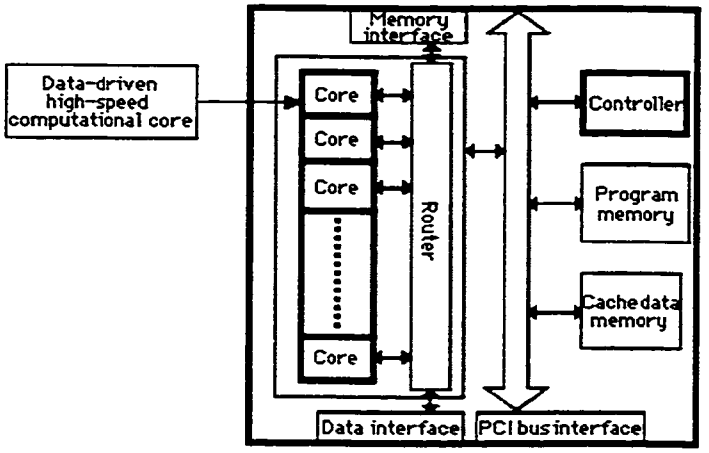


Figure 26 - Block Diagram of DDMP

4.6.6 Pentium processor with MMX technology

The motivation behind MMX[27] is to provide additional capability to existing processors without sacrificing backward compatibility. It has been added to existing floating point and integer functional units as shown in Figure 27.

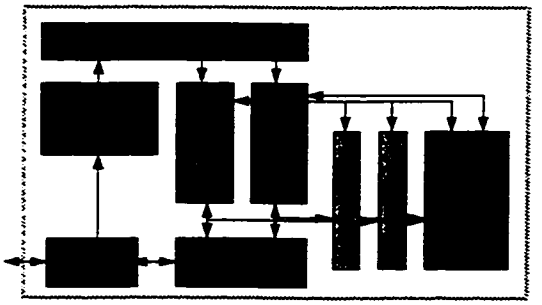


Figure 27 - Implementation of MMX technology

MMX technology processes several pieces of data with each instruction. Typical elements of data are usually small, for example 8 bits for each pixel color component in an image or 16 bits per element for audio samples. CPU Data in MMX technology are wide (i.e. 64 bits or more) and are composed of independent smaller size data elements called packed data types. A rich set of MMX instructions are defined to perform the parallel operations

on multiple data elements packed into new wide data types (for example 8 x 8-bit, 4 x 16-bit). Processor extends the basic integer instructions into SIMD versions. MMX instructions also support saturating arithmetic in which the overflow and underflow bit is not truncated and the instruction results in the largest or smallest possible representable number in the data type of operation. Sub-word parallelism on packed data types and saturation arithmetic in MMX technology are useful in many multimedia applications such as motion compensation and graphics algorithms like shading. MMX technology also provides a parallel compare instruction for data dependent applications. In Intel Pentium processors with MMX technology, MMX instructions are designed to run in the integer pipelines of the CPU despite the use of the floating point registers to hold data. MMX instructions with the exception of the multiply instructions execute in one cycle. The multiply instructions have an execution latency of three cycles, but the multiply unit's pipelined design enables a new multiply instruction to start every cycle.

4.6.7 C-Cube's VideoRISC Processor (VRP)

The VideoRISC [81] family consists of a series of video compression products for digital television, consumer electronics, and multimedia computing applications. VideoRISC products are a combination of micro-application software sets and microprocessor chips. A different micro-application is supplied for each product and defines the functionality of that product. For example, the CLM4500 is a real time MPEG-1 video encoder (for consumer quality), while the CLM4200 is a real time H.261 video codec. Both the CLM4500 and CLM4200 processors are based on VideoRISC product.

While each micro-application is different, all run on the same chip: C-Cube's VideoRISC Processor (VRP). The VRP is designed to compress and decompress digital video in real time, and can be used individually or with other VRPs, depending on the performance requirements of the micro-application. The CLM4600 MPEG-1 Video Encoder (for broadcast quality) requires eight VRPs, while the CLM4500 requires only two.

Other members of the VideoRISC family include the desk-top-oriented CLM4100 Multimedia Accelerator and an MPEG-2 encoder.

As an example of this family, CLM4700 MPEG-2 digital video encoder chip-set [30] encodes broadcast-resolution video into MPEG-2 Main Level/Main Profile format in real time, using either frame encoding or adaptive field/frame encoding techniques. System features include:

- MPEG-2 Encoding
- Multi-resolution /Multi-mode Video Capability
- System Layer Support
- Support for Broadcast Applications
- Simplified Hardware Architecture

4.6.8 L64002 MPEG Audio/Video Decoder

L64002 is a single-chip MPEG-2 source decoder [82] that combines a video decoder that is compliant to the MPEG standard Main Profile at Main Level with a two-channel MPEG audio decoder. The L64002, however is more than just a single chip MPEG-2

audio/video source decoder. The architectural elements of the device shown in Figure 28 were developed for implementation of compressed digital interactive television applications. These architectural elements include a customized RISC engine and a video display and graphics controller. Features of L64002 include:

- Audio Decoding Block
- Decodes Layer I and Layer II (MUSICAM) ISO 11172
- Decodes two channels of 5.1 channel bit-stream (ISO 13818)
- Output samples rates: 16, 22.05, 24, 32, 44.1, 48 kHz
- Channel data rates of 8 KBits/sec to 448 KBits/sec
- Outputs 16-bit PCM audio
- Customized RISC Engine
- All microcode stored on-chip
- Serial or 8-bit parallel input
- Robust error concealment
- Checks for syntax errors at all layers of MPEG bit-stream
- Freeze frame for video; mute or repeat for audio
- Optimized Memory Architecture

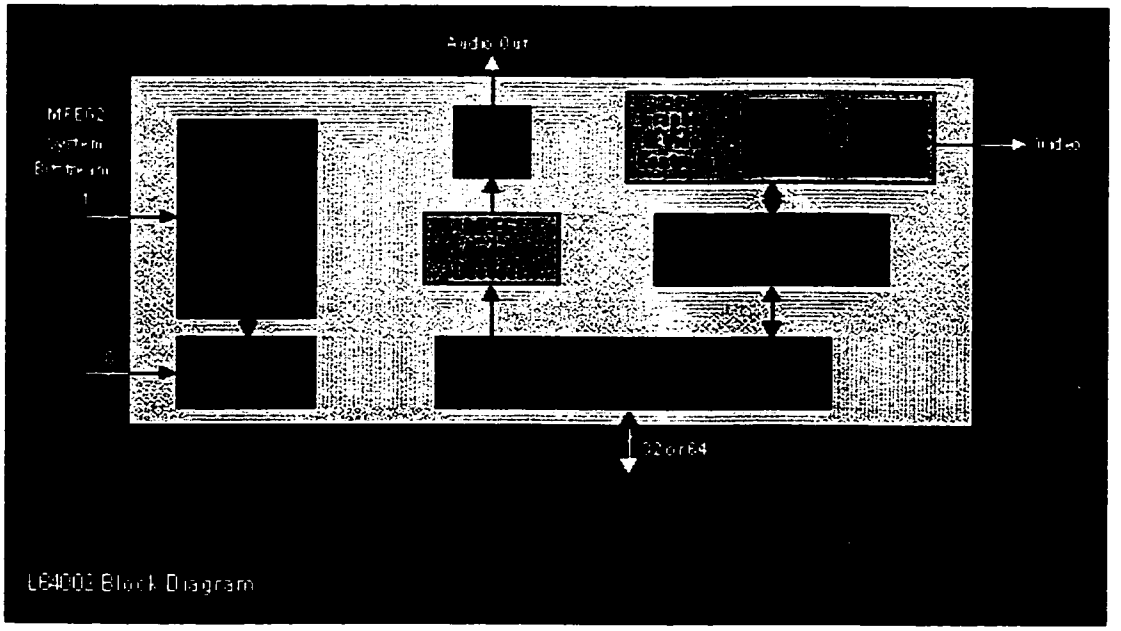


Figure 28- L64002 Block Diagram

4.6.9 IBM Video Integration Processor

IBM introduced Video Integration Processor (ViP905) [83] on a single 208-pin PQFP module. ViP905 is designed in CMOS 0.5 micron, triple level metals, contains over 750,000 transistors, 250,000 gates, and provides 900 million operations per second. This technology provides the ability to process a television-like RGB or YUV data stream from a TV digitizer function or video CODEC (either Software [S/W] or Hardware [H/W]) into computer memory for manipulation and display. The image can be scaled to any desired size, from one pixel to four times (4X) the size of the original—in full motion, on the fly. The extremely sophisticated scaling algorithms provide high-quality images, without the artifacts introduced by other methods. The TV data stream can be transformed into RGB24, RGB16, or RGB8 screen formats. Proprietary Dithering

Algorithms improve the quality of RGB16 output to approximately RGB24 quality, and improve the quality of RGB8 output to approximately RGB16 quality.

The block diagram of ViP905 is shown in Figure 29. The Video Integration Processor technology is capable of 60-Hz interlaced updates of the TV decoder video streams, or 30-Hz non-interlaced updates of both the TV decoder stream and the video CODEC stream. Two video windows can overlay each other, as desired, with single pixel granularity. In addition, graphics can be overlaid on the video windows with single pixel granularity.

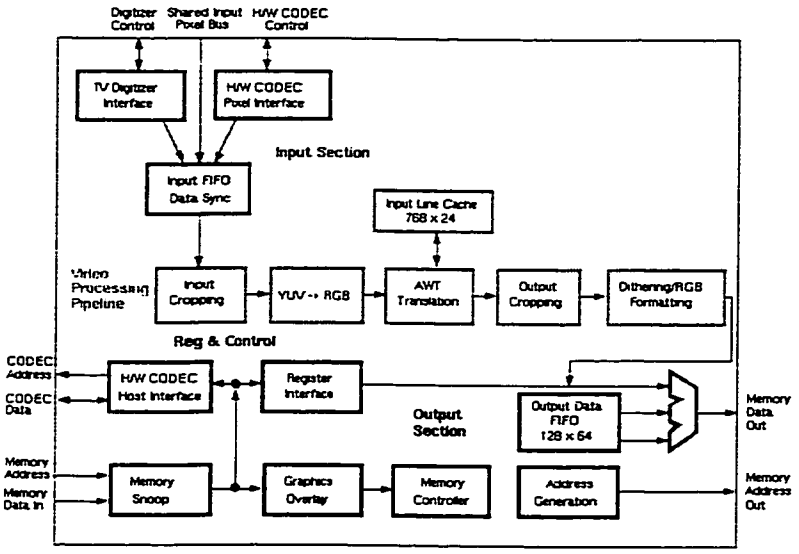


Figure 29- VIP Block Diagram

4.6.10 8x8's Video Communication Processor (VCP)

The 8x8 Video Communications Processor (VCP) is a single-chip programmable video subsystem and multimedia communications processor [84]. It can implement a complete multimedia and video conferencing subsystem on a single circuit card with a

programmable DSP chip and memory. The VCP performs a superset of the functions of 8x8's Vision Controller and Vision Processor chips. For video conferencing applications it can act as a full CIF resolution H.261 codec and provide forward error correction and bit-stream multiplexing to the H.221 and H.242 standards. For video playback applications the VCP can decode the MPEG-1 video and audio streams. In addition to multiplexing and codec functions, the VCP provides programmable video pre- and post-processing functions including format conversion, video scaling, temporal filtering, output interpolation, color conversion and picture-in-picture.

4.6.11 Array Microsystems Video Compression Chip-set

Array Microsystems designed a two chip chip-set [85] for video compression applications. The a77100 Image Compression Coprocessor (ICC) and a77300 Motion Estimation Coprocessor (MEC) chip-set provides a programmable video compression solution with reasonable performance and features for multimedia systems.

The ICC performs functions such as DCT, quantization, zero-run length coding, etc. The MEC performs motion estimation and is required only in those systems implementing MPEG-1 or H.261 motion compensated compression. The block diagrams of ICC and MEC are shown in Figure 30. For increased flexibility, the host PC or an off-the-shelf RISC microcontroller performs variable length coding and bit-stream control, communicating with the ICC and MEC over their respective host bus interfaces. Input, output, and scratchpad images are stored in DRAMs or VRAMs connected to the ICC and MEC video buses. This memory for example, supports the following at 30 fps:

- JPEG encoding or decoding of full resolution CCIR-601 (720h x 480v) images

- Simultaneous H.261 encoding and decoding of CIF (352h x 288v) images
- Full MPEG-1 I,B,P encoding or decoding of SIF (352h x 240v) images.

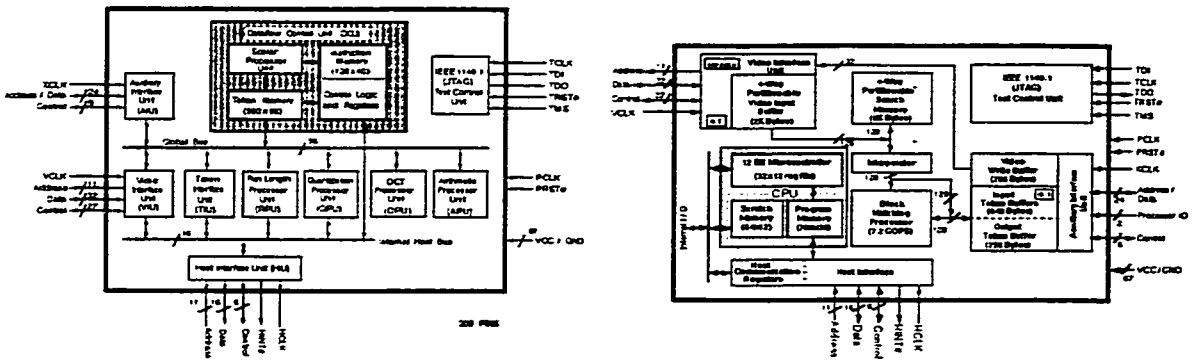


Figure 30- ICC and MEC block diagram

In the next section we present an analysis of the strengths of each processor in order to create a model for an ideal multimedia processor. These ideas are considered in designing the Fractal Engine.

4.7 Analysis

We now present the appropriate architectural solutions for multimedia applications based on the analysis of multimedia data and processing as well as the analysis of architectural approaches. We note that the designer has to decide upon the critical options based on available VLSI technology, target application and environment.

- Multimedia processing and high throughput CPUs are employed not only in desktop computing applications to enhance the computing power of advanced workstations and servers but also in many embedded applications such as high-speed printers and video game consoles. Hence:

⇒ There is no unique solution for all multimedia systems.

- Most low and medium level algorithms have pre-determined memory access. Hence:

  ⇒ Partitioned memory architecture among data paths and a shared memory architecture is sufficient for those operations (in contrast to complex multi-port memories).

- Real-time processing is stream based and has poor temporal locality. Hence:

  ⇒ The increased number of data cache misses coupled with the high communication bandwidth between cache and register file degrades the system performance. However, block transfer operations speeds up the entire process.

- High throughput memory interfaces are required to maintain all the functional units busy all the time. Hence:

  ⇒ DMA interfaces are employed in multimedia processors. For example Mpact 2 has DMA interfaces.

  ⇒ Rambus interface is more appropriate for data transfer. Therefore, V830R implements RDRAM interface.

  ⇒ On chip caches with multiple ports for simultaneous read and write increase data bandwidth. Mpact 2 has a data cache with 6 ports for reading and for writing.

  ⇒ State-of-the-art bus interfaces such as AGP, PCI should be implemented.

  ⇒ Utilization of wide CPU words (i.e. 64 bit word) and data buses result in an increase in data throughput.

- The MAC operation is very common. Hence:

  ⇒ DSP arithmetic units are appropriate.

- The conditional branch is not used very frequently. Hence:

  ⇒ Superscalar, VLIW and pipeline architectures work well.

- The operations have inherently high parallelism. Hence:

  ⇒ Compilers for VLIW processors extract the parallelism and generate efficient code. This is the reason for most media processors such as Mpact 2 and TriMedia being based on VLIW architecture.

- There are high level and medium level applications in multimedia processing which require increased compute power from the processor (in the range of million operations per second) such as affine transformations, motion estimation and 3D rendering. Hence:

  ⇒ hardware dedicated units are required. For example, Mpact 2 has a motion estimation unit and an engine for 3D rendering. TriMedia has a variable length decoder for MPEG decoding and a scaling unit for video post processing.

- There are conditional execution of instructions in multimedia algorithms. Hence:

  ⇒ MIMD control structure enables each individual data path to adapt its execution path accordingly. This approach is employed in MVP.

- Most of the multimedia functions don't require more precision than 8 or 16 bits. Hence:

⇒ fine to medium grain architectures are more suitable.

⇒ Packed arithmetic is employed for concurrent execution of packed data in wide data words like the parallelism in MMX technology for Pentium processors.

- Floating point operations are commonly used in applications like 3D graphics. Hence:

  ⇒ Floating point units speedup the execution of these operations at the expense of additional real estate in the chip as in the case of MVP and Pentium processors.

- Concurrent execution on sub-words of data is possible in multimedia instructions especially in wide data words (i.e. 64 bit)

  ⇒ Multimedia extensions to individual instructions are justified to exploit sub-word parallelism. This approach is employed in Pentium processors with MMX technology.

- Conditional branches which alter the path of execution and reduces pipeline utilization are present in data dependant applications such as object recognition, video compression and model-based representation. Hence:

  ⇒ Out-of-order execution and dynamic scheduling techniques which can be incorporated in super-scalar architectures such as Pentium processors, are used to enhance utilization factor of pipelines.

- High speed access rate are desired in multimedia processors to speedup the operations. This results in high frequency clock rates and therefore result in increased power consumption. Hence:

⇒ Data flow processors like DDMP without a clock signal decrease the power consumption drastically.

4.8 Summary

Multimedia hardware architectures have evolved from simple extensions of digital signal processors and small dedicated architectures to powerful parallel architectures. It is necessary for the designer to investigate the various issues in this evolution before embarking on a new hardware design.

In this chapter, first the issue of programmability has been studied. Different techniques and approaches ranging from dedicated modules to full programmable media processors have been presented. Based on available VLSI technologies, required computational achievement, production quantity and the target algorithm, it is the designer who will select the best VLSI implementation approach. In our proposed Fractal Engine we employ both techniques in different modules. Critical hardware units are implemented in maximum efficiency. Complex multimedia processing tasks namely affine transforms are directly mapped to these units. Control unit and other programmable units are implemented using configurable FPGAs. Programmability feature exists in Fractal Engine by communication with an external CPU which controls the operation of Fractal Engine.

Programmable processors for multimedia applications are increasingly becoming popular due to the wide variety of multimedia applications, development of multimedia technology, advancements in parallel processing techniques, availability of high speed interconnection networks and memories and enabling VLSI technology. In this chapter,

various aspects of a programmable multimedia processor have been presented. Finally, different examples of available processors have been studied. The features of recent programmable multimedia processors are summarized and tabulated in Table 1. We note that although TI-MVP and Pentium-MMX are older than other architectures, they are included in the table because of their high performance and representation of advanced techniques..

	TI – MVP	TriMedia	Mpact	MDMP	V830R	MMX
Processor	RISC,DSP, CISC	RISC	RISC		RISC, DSP	CRISC
Granularity	Medium	Fine-Medium	Medium	Medium	Medium	Medium
Parallelism	LIW	VLIW	VLIW	Dataflow	Superscalar	Superscalar
Data Distribution	MIMD	SIMD	SIMD	MIMD-like	SIMD	SIMD
Memory	VRAM	SDRAM	RDRAM		RDRAM	EDO , SDRAM

Table 1 - Features of multimedia processors.

In the next chapter we start the design of Affine transform processor which is the core of the Fractal Engine. Affine transforms are first presented followed by derivation of two fundamental affine transforms. The hardware implementation is discussed in the end of the chapter.

5 Affine Transform Processor

The core processing element for Affine processing is *Affine Transform Processor* (ATP) which is a parallel and pipelined architecture. ATP is simple, modular, scaleable and is optimized to execute both low level and mid level operations. Implementation of the basic operations by ATP enables efficient execution of a majority of visual computing tasks. ATP executes Affine transforms which are a geometric transformation.

The basis of geometric transformations[98] is the mapping of one coordinate system onto another. This is defined by means of a spatial transformation (a mapping function that establishes a spatial correspondence between all points in the input and output images). With a spatial transformation, each point in the output image (x, y coordinates) maintains the intensity value of its corresponding point in the input image (u, v coordinates). The correspondence is found using the spatial transformation mapping function ($X(u,v)$, $Y(u,v)$) to project the output point onto the input image. Figure 31 illustrates a typical transformation.

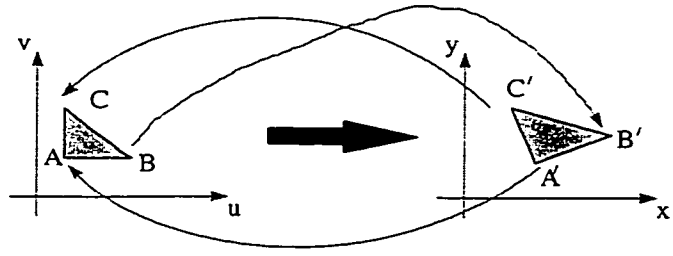


Figure 31- Spatial Transformation.

We note that in the Figure 31, the intensity values of the pixels are the same in the input and output images. Depending on the application, spatial transformation mapping functions may take on many different forms. Simple transformations may be specified by analytic expressions including affine, projective, bilinear and polynomial transformations.

Affine transforms[39] are widely used in visual processing applications. A description of affine transforms and derivation of the two fundamental operations are presented in the next sections followed by an efficient method for implementing the two basic operations which form the core of the proposed ATP.

5.1 Affine Transforms

Affine (linear) transforms, specified by analytic expression as a matrix multiplication, are the most commonly used spatial transform in the area of image and video processing. They map a 2-dimensional Euclidean space R^2 onto itself as shown in Figure 32. Affine mappings preserve existing parallelism (lines) in the original image. For affine transformations the mapping functions are:

$$\begin{aligned} x &= a_{11}u + a_{12}v + a_{13} \\ y &= a_{21}u + a_{22}v + a_{23} \end{aligned} \leftrightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix} \quad (1)$$

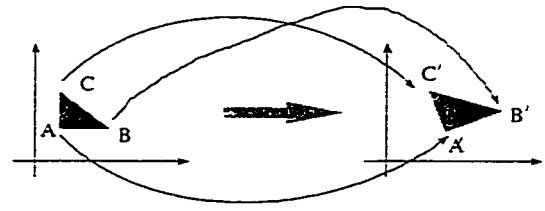


Figure 32 - General Affine Transformation

This accommodates translations, rotations, scale, and shear. Affine transformation is also expressed using a *3x3* matrix for homogenous coordinates.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

We note that the combinations of two consecutive affine transforms are easily expressed by the product of their individual transform matrices (i.e. it is another affine transform). It is also shown that any arbitrary affine transform can be expressed as a set of predefined affine transforms, which include translation, scaling, shear, transposition and rotation.

5.1.1 Translation

All points are translated to new positions by adding offsets T_u and T_v to u and v, respectively. The translated transform is expressed in Equation (3) and is illustrated in Figure 33.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_u \\ 0 & 1 & T_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

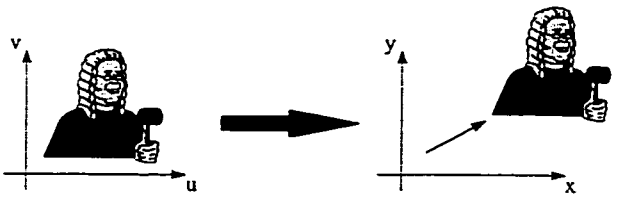


Figure 33- Translation.

5.1.2 Scale

All points are scaled by applying the scale factors S_u and S_v to the u and v coordinates, respectively (Equation (4)).

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S_u & 0 & 0 \\ 0 & S_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

If the scale factors are not identical, then the image proportions are altered resulting in a disproportionate scaled image. Positive scale factors that are larger than unity result in magnification while factors smaller than unity result in a reduction. Negative scale factors cause the image to be reflected. An example of positive scaling is shown in Figure 34.

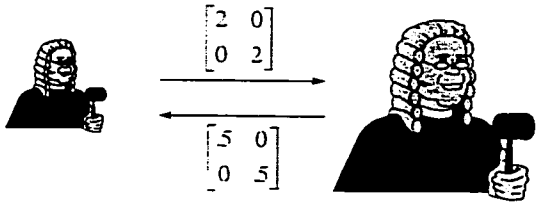


Figure 34- Scale.

5.1.3 Shear

By allowing a_{12} to be non-zero, x is made linearly dependant on both u and v, while y remains identical to v. A similar operation can be applied along the v-axis to compute the new values for y while x remains unaffected. This effect is called shear. The shear transform along the u-axis and v-axis are as follows:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & H_v & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad \text{Or} \quad \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ H_u & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{5}$$

An example of shear along x-axis is illustrated in Figure 35.

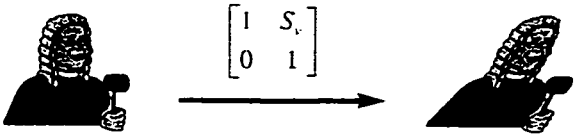


Figure 35- Shear.

5.1.4 Transposition

All points in the uv-plane are reflected so that the x-coordinate will correspond to v and y-coordinate to u.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

An example of transposition is shown in Figure 36.

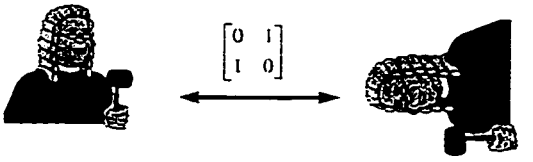


Figure 36- Transposition.

5.1.5 Rotation

All points in the *uv*-plane are rotated about the origin through a counterclockwise angle θ. The transform matrix is given in (7).

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Each point in the image is rotated, so that the distance of the point from the origin is a constant (as shown in.

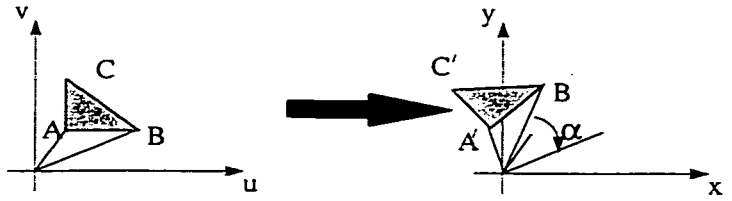


Figure 37- Rotation procedure.

An example of 45-degree rotation is illustrated in Figure 38.

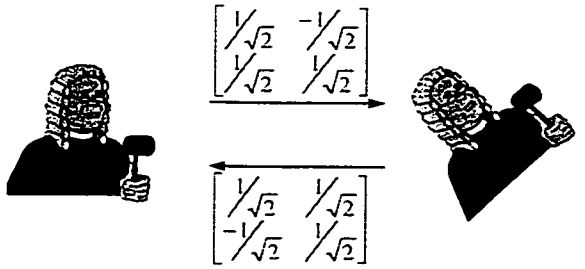


Figure 38- Rotation.

The inverse of a rotation is also a rotation with the same degree but in the opposite direction and can be simply expressed as:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Implementation of Rotation in Digital Domain is discussed in Chapter 7.

5.2 Fundamental Affine Operations

A set of special affine transforms typically used in several image and video processing applications are applied on intensity values of a square block of pixels (L=MxM pixels). We denote these operations by A_{st}^{k} which consist of stretching (s), translation (t) and isometric transform (k).

If $\mathbf{X}$ is an L-dimensional vector, then

$$\begin{aligned} &A_{st}^{k}(\mathbf{X}) = sI^{k}(\mathbf{X}) + tI^{*} \\ &s > 0; \end{aligned} \tag{1}$$

where s,t are integers from the sets S, T and define stretching and translation mappings.

$$S = \{S_s; s = 1,2,\cdots,N_s\}$$
$$T = \{T_t; t = 1,2,\cdots,N_t\}$$

I^{k} is one of the isometric transforms given in the set I,

$$I = \{I^{k}; k = 1,2,\cdots,N_k\}$$

and I^{*} is an L-dimensional identity vector = $[1,1,\cdots,1]$.

The following basic isometric transformations have been chosen among all isometric transforms[41].

I_1 - Identity: This transform maps each pixel onto itself.

I_2 - Reflection about the mid-vertical line: Each pixel with (x,y) coordinates is mapped onto a pixel with $(-x,y)$ coordinates.

I_3 - Reflection about the mid-horizontal line: Each pixel with (x,y) coordinates is mapped onto a pixel with $(x,-y)$ coordinates.

I_4 - Reflection about the first diagonal: This affine transformation swaps the coordinates of each pixel and is also called transposition. A pixel with (x,y) coordinates is mapped onto (y,x).

I_5 - Reflection about the second diagonal: This isometric transform swaps the coordinates of each pixel and also changes the sign of the values of coordinates. A pixel with (x,y) coordinates is thus mapped onto $(-y,-x)$.

I_6 - Rotation around the center by 90 degrees: This transformation rotates the picture 90 degrees to the left (counter clockwise). A pixel with (x,y) coordinates is hence mapped onto $(y,-x)$.

I_7 - Rotation around the center by 180 degrees: Each pixel in this transformation is reflected about the center of the picture. A pixel with (x,y) coordinates is mapped onto $(-x,-y)$.

I_8 - Rotation around the center by 270 degrees: This transformation rotates the pictures 90 degrees to the right (clockwise). A pixel with (x,y) coordinates is thus mapped onto $(-y,x)$.

An example of the mapping of selected affine transforms is illustrated in Figure 39.

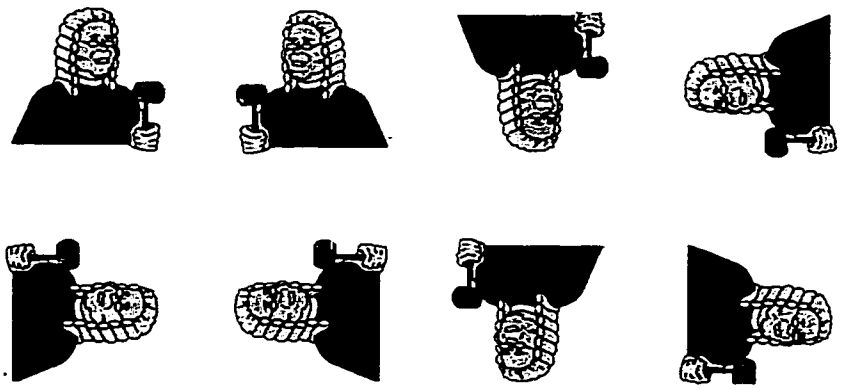


Figure 39 - Example of isometric transforms.

We note that the combination of any pair of these transforms will result in another transform from this set. For example, a reflection about the mid-horizontal line (I_3) followed by a rotation around the center by 90 degrees (I_6) will result in a reflection about the second diagonal (I_5) (i.e. $I_6 \circ I_3 = I_5$). Table 2 lists all possible combinations in I.

Table 2

comb. (○)	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
I_1	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
I_2	I_2	I_1	I_7	I_6	I_8	I_4	I_3	I_5
I_3	I_3	I_7	I_1	I_8	I_6	$\textcircled{I_5}$	I_2	I_4
I_4	I_4	I_8	I_6	I_1	I_7	I_3	I_5	I_2
I_5	I_5	I_6	I_8	I_7	I_1	I_2	I_4	I_3
I_6	I_6	I_5	I_4	I_2	I_3	I_7	I_8	I_1
I_7	I_7	I_3	I_2	I_5	I_4	I_8	I_1	I_6
I_8	I_8	I_4	I_5	I_3	I_2	I_1	I_6	I_7

We propose to employ a chain of combinations of two simple isometric transforms, I_3 (Reflection about the mid-horizontal line) and I_4 (Reflection about the first diagonal or Transposition) to express all other transforms as follows.

$$I_3 = I_3$$
$$I_4 \circ I_3 = I_8$$
$$I_3 \circ I_4 \circ I_3 = I_5$$
$$I_4 \circ I_3 \circ I_4 \circ I_3 = I_7$$
$$I_3 \circ I_4 \circ I_3 \circ I_4 \circ I_3 = I_2$$
$$I_4 \circ I_3 \circ I_4 \circ I_3 \circ I_4 \circ I_3 = I_6$$
$$I_3 \circ I_4 \circ I_3 \circ I_4 \circ I_3 \circ I_4 \circ I_3 = I_4$$
$$I_4 \circ I_3 \circ I_4 \circ I_3 \circ I_4 \circ I_3 \circ I_4 \circ I_3 = I_1$$

Hence, the two fundamental operations in selected affine transforms are transposition and reflection about mid-horizontal line. This implies that the implementation of these two transforms in a chain will result in all other transforms without explicitly implementing them.

5.3 VLSI Implementation of ATP

This module is capable of executing for each range block, all of the selected isometric transforms on the domain blocks and selects the best transform corresponding to the closest match. The basic isometric transforms are transposition and reflection. Hence, a chain of these fundamental operations is implemented in **AFM** (Affine Module) in order to execute all of the selected transforms in a systolic fashion as shown in Figure 40.

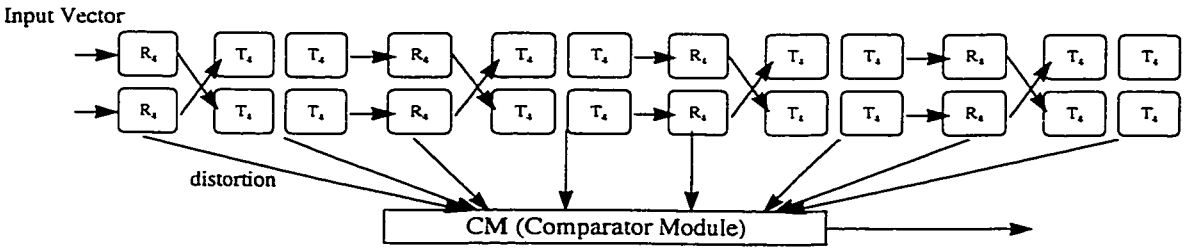


Figure 40- Affine Module Block Diagram

Every unit in the chain has a built-in array adder (AR) and distance calculator (D) to measure the distance between the transformed domain block and the range block stored in the SRAM. The design of built-in array adder is detailed next.

5.3.1 Array Adder Unit (AR)

This module consists of two sets of *M* basic cells, where *M* is the number of rows or columns of the input block (a *4x4* example *is shown in* Figure 42). The first basic cell (accumulator (a_i) shown in Figure 41) accumulates the partial distortion for the *i*th row of the domain block with the corresponding row of the range block stored in the **SRAM** ($\sum_j \left| r_{ij} - (sd_{ij} + t) \right|$). In the first *M* clock cycles, the absolute value of the difference between the row elements of the domain block and corresponding elements of the range block is accumulated. At the end of every *M* clock cycles, the accumulated value (S_i) is ready to be output.

The second set of basic cells (summation, s_i) adds the partial distortion values to compute the total distortion value. The block diagram of the cell is illustrated in Figure 41.

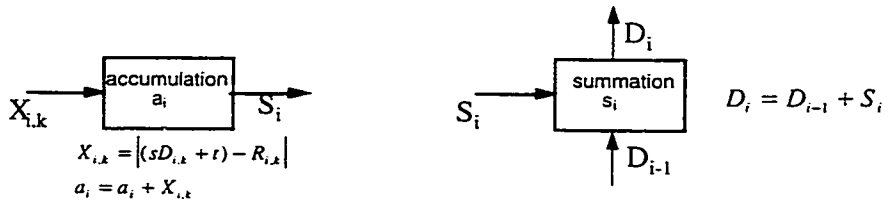


Figure 41- Accumulation and Summation Cells

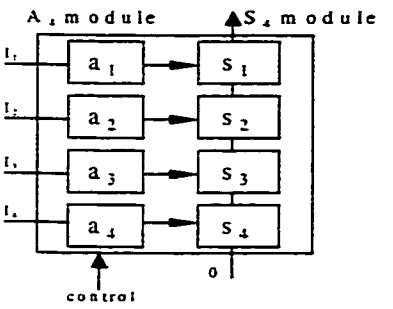


Figure 42- Array Adder for *4x4* blocks

5.3.2 Reflector Unit

The reflector unit shuffles the input columns of the data such that:

$$Y_{ij} = X_{(M-j+1)j} \qquad \text{for } i = 1,2,\ldots,M$$

Where X_{ij} is the input to the module and Y_{ij} is the output. This module delays the output for M clock cycles to maintain the synchronization between the outputs of other modules.

5.3.3 Transposer Unit

A parallel and pipelined transposer architecture was proposed in [88]. Here, the basic cell of the transposer architecture (as shown in Figure 43) has two modes of operation A and B selected by a control signal C such that when

1. C=1 A OUTPUT = A INPUT (A mode)
2. C=0 B OUTPUT = B INPUT (B mode)

We note that **b** indicates the data-bus width. The control signal is derived from the global clock signal. Thus the communication is synchronous and the control is simple in structure.

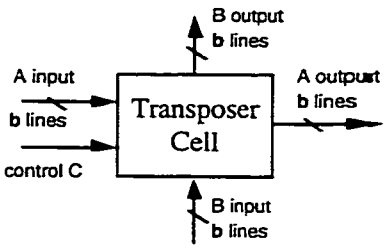


Figure 43-Basic Transposer Cell

The unit also includes an array adder. The architecture of a transposer module is shown in Figure 44. This module consists of $L=M^2$ basic cells. Figure 44 illustrates the design of transposer for a $4x4$ matrix (i.e. $M=4$).

An entire column (row) is loaded in and out of the module in each clock cycle. In the A-mode, a column (initially the first column) of a block is loaded in parallel into the cells $T_{1,1}$-$T_{1,4}$. Meanwhile, the second column of data is prepared to be loaded into the transposer module. In the second clock cycle, they are loaded into the cells $T_{1,1}$-$T_{1,4}$, while the first column of data moves to the cells $T_{2,1}$-$T_{2,4}$. This procedure continues and at the end of *4* clock cycles, all the columns of data are loaded into the transposer module. As soon as the last column of data is loaded into the cells $T_{4,1}$-$T_{4,4}$, the cells are switched to the B-mode of operation. In the next *4* clock cycles, the row elements of the input block are drawn out of the transposer module through the outputs B_1 -B_4 in the B-mode. Note that this output data is essentially the transposed version of the input data.

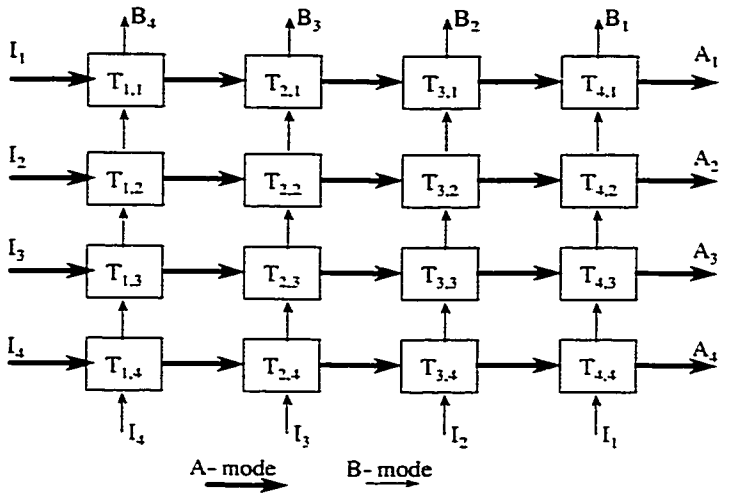


Figure 44- *4x4* Transposer Module

5.4 Summary

We reiterate that the ATP forms the core of the Fractal Engine architecture. Considerable optimization has been applied in the design of ATP in VHDL. The derivation of two fundamental affine transforms and the design of array adder units are the key factors that lead to a highly parallel, pipelined and scalable architecture of ATP. The proposed architecture for ATP is scalable and modular and is hence suitable for VLSI implementation. In the next chapter, the design of the Fractal Engine and its associated peripheral blocks are presented.

6 Fractal Engine

The primary focus of the Fractal Engine is to implement both image and video based algorithms and it is based on the affine processor core. In this chapter, fractal processing algorithm is first introduced followed by the design of Fractal Engine. Example algorithms from spatial domain where image and intra-frame calculations are considered and temporal domain, where temporal correlations are exploited are presented. The spatial and temporal operations are mapped onto the Fractal Engine. Finally, timing analysis demonstrates the real-time execution potential of the algorithms using the Fractal Engine.

6.1 Why Fractal?

We recall from chapter 2, that the choices of kernels used in our design were primarily dictated by visual data processing requirements. We note that a majority of low level and mid level visual data processing exists in fractal block processing (FBP). Fractals exploit the high correlation and self-similarities present in the visual data within an image or a sequence of images. Fractal processing extracts existing self-similarity and self-affine within an image.

FBP encompasses a majority of image processing operations including, summation / accumulation, image addition / subtraction, translation, stretching, shifting, scaling, rotation and pattern matching. We have therefore chosen FBP as the candidate algorithm for the design of the generic video processing element in the proposed architecture. Furthermore, from the current trends in multimedia design, model based representation,

complex motion analysis and image understanding are the most demanding tasks. These will form the major requirements for a machine to interact meaningfully with its environment. Hence, affine transforms are receiving increasing attention in recent research including the MPEG4 standard. We note that fractal block processing is about finding affine relations within the blocks or objects of an image and is an appropriate candidate for future visual data processing applications. A detailed description of FBP is presented in the following section.

6.2 Fractal Block Processing

The emergence of powerful hardware architectures is providing the possibility of using FBP in image and video processing. Fractal based techniques are becoming increasingly popular in visual processing. They have been applied in several areas of visual processing, such as segmentation[47], analysis[48], [49], synthesis

[50], computer graphics[55] and compression[56], [57]. In the last few years, several image compression methods using fractal theory have been developed. These methods promise better compression performance. Since fractal images can be described and generated by simple recursive mathematical equations operating on the entire image, the basic idea is that an image can be reconstructed based on the self-similarity it contains. During the analysis stage, the algorithm partitions the image into a number of square blocks. For each block, FBP associates the transformation in the image, which can best reconstruct the block. This information can be used in different areas. For instance, in image coding, compression is obtained by storing only the description of these

transformations. Expected compression ratios for moderate quality reconstruction are about 100:1. Fractal processing offers the following advantages and strengths:

1. Due to the existing self similarity in many parts of natural images, fractal processing is suited for real world pictures.
2. The degree of analysis can be traded off against processing time.
3. After analyzing the image, reconstruction is very fast.
4. It provides scalability/resolution independence since the image is defined by a set of equations which can be arbitrarily scaled.

However, fractal processing has the following disadvantages:

1. Most natural images are not mathematically synthesizable. They are not self-transformable at the level of the entire image.
2. The procedure to calculate and exploit the existing correlations within an image or images is highly compute intensive which precludes real-time implementation of fractal based algorithms.

To overcome the first problem, Fractal Block Processing (FBP) has been proposed in the literature [41]. FBP assumes that visual correlation can be efficiently exploited through piecewise self-transformability on a block-wise basis. The image is partitioned into non-overlapping blocks called range blocks. For each range block, possible affine contractive transforms are applied on all candidate (domain) blocks within the image. The goal is to find the best match domain block for every range block. At the first level, larger range blocks (typically *32x32*) and larger domain blocks are considered. If a range block cannot

be approximated (within a given threshold) by the domain blocks in the image, it is further divided into smaller size range blocks in the next level and the best match search is repeated. We note that this technique is based on Partial Iterated Function Systems (PIFS), in which the image is expressed using several equations and mappings.

The key element in FBP is affine contractive transforms. They are linear transforms which map a 2-dimensional Euclidean space R^2 onto itself and are described as follows (The detail of these transformations and an Affine Processor are presented in chapter 5) :

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} + \begin{bmatrix} E \\ F \end{bmatrix}, \qquad \text{where} \quad \left\| \begin{bmatrix} A & B \\ C & D \end{bmatrix} \right\| < 1 \tag{9}$$

This indicates that the image will be formed of properly transformed parts of itself. The goal is to find the best set of affine transforms (W) which minimize the distortion between the transformed image ($W(f)$) and original image (f).

W is a collection of maps w_i identical to a pair of a range block and a domain block and the parameters of the corresponding affine transform.

$$W = \bigcup_i w_i \quad \text{i.e.} \quad W(f) = w_1(f) \cup w_2(f) \ldots \cup w_N(f) \quad \text{and } f \text{ is as close as possible to } W(f) \tag{10}$$

The execution time for QCIF (180x144), CIF (360x288) and CCIR 601 (720x480) video sequences corresponding to a 100MHz clock (with the assumption that one operation is executed every clock cycle) are 6.35, 101 and 1000 seconds, respectively. Hence, a speedup factor ranging from 190 to 30000 is required for real-time processing. There are

two basic operations involved in FBP namely affine transformation (discussed in chapter 5) and mean/variance computation, which is now presented.

6.2.1 Mean and Variance computation

To normalize each domain block before comparison with a range block, mean and variance values of the blocks are calculated. These two mathematical entities are the basis of all statistical operations in image processing and are expressed as follows:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{L} x_i \quad , \sigma^2 = \frac{1}{N}\sum_{i=1}^{L}(x_i - \bar{x})^2 \qquad (11)$$

We note that in these expressions, the calculation of mean value and variance of the block are executed serially. However, the expressions in (3) can be rewritten for parallel execution.

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{L} x_i \quad , \sigma^2 = \frac{N\sum x^2 - (\sum x)^2}{N^2} \qquad (12)$$

We note that the fundamental operations involved in (4) are squaring, division and accumulation which are implemented in a dedicated hardware unit in the Fractal Engine.

In the following section, the different modules of Fractal Engine are presented.

6.3 Fractal Engine

We propose a parallel and pipelined architecture based on ATP core called **Fractal Engine** to implement the operations in FBP. Fractal Engine is simple, modular, scalable and is optimized to execute both low level and mid level operations. We present the

design of individual sections of the Fractal Engine – Dedicated module which is shown as processing section in Figure 45.

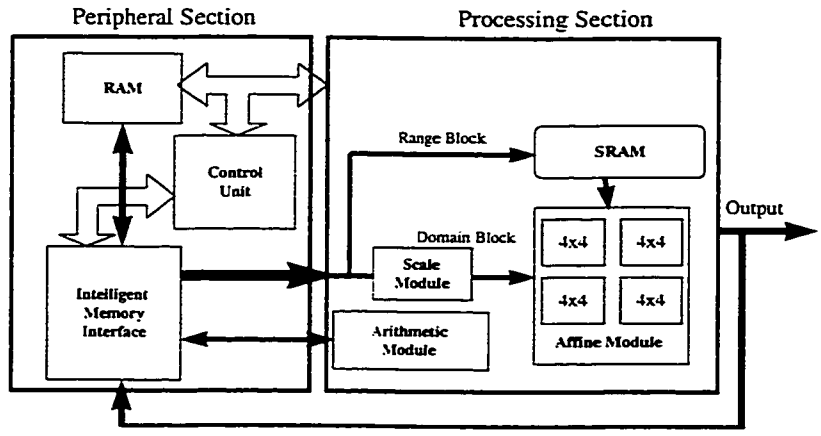


Figure 45- Fractal Engine Block Diagram

6.3.1 Processing Section

This unit essentially forms the dedicated module of Fractal Engine and performs all calculations required in FBP. It consists of three modules:

- Affine Module (AFM): to execute isometric transforms and calculate the distortion between the range and domain blocks.
- Scale Module (SCM): to execute scaling and translation.
- Arithmetic Module (ARM): to calculate the mean and variance of blocks.

In addition, the processing section has a built-in static RAM (SRAM) to store range blocks.

6.3.1.1 Affine module (AFM)

This is the core processor of Fractal Engine detailed in chapter 5.

6.3.1.2 Scale module (SCM)

The task of this module is to calculate the translated and scaled version of every domain block and make it available for geometric transformations in AFM. Several parallel units are implemented to execute scaling and translation on different domain blocks in parallel.

6.3.1.3 Arithmetic module (ARM)

This module executes low level computing operations to calculate the mean and variance of image blocks. One element of a block is pumped into the module at every clock cycle.

6.3.2 Scalability

Hardware scalability is an important feature in the design of an architecture. For a problem of complexity X which is executed using N units in T seconds, scalability implies:

- T/M seconds will be required to solve the problem using NM units.
- A problem of complexity XM is solved in T seconds using NM units.

The first type of scalability requires a flexible control design, while the second type of scalability requires that the feature of scalability be incorporated in the design of individual modules. We illustrate the concept of scalability in Fractal Engine, where *8x8* block architectures have been built using *4x4* blocks.

6.3.2.1 Scalable array adder

An 8-element array adder is built using two 4-element array adders as shown in Figure 46.

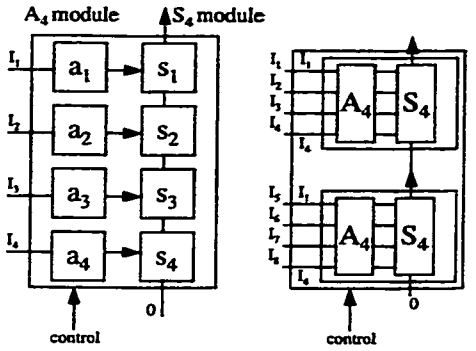


Figure 46- *4*-element and *8*-element Array Adder

6.3.2.2 Scalable reflector

The procedure of reflection about mid horizontal line is straight forward. This module only shuffles the input elements entering the module. We note that an R_8 module can be configured using two R_4 modules as shown in Figure 47.

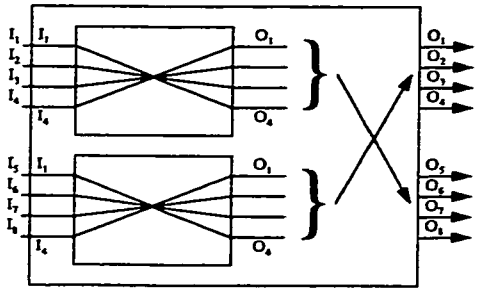


Figure 47- Reflector Module

6.3.2.3 Scalable transposer

The proposed transposer is a modular and scalable architecture. To build an *8x8* matrix transposer, we simply arrange four 4x4 matrix transposers together as shown in Figure 48. The transposition process is executed in 8 clock cycles. The only modification required is in the frequency of the control signal.

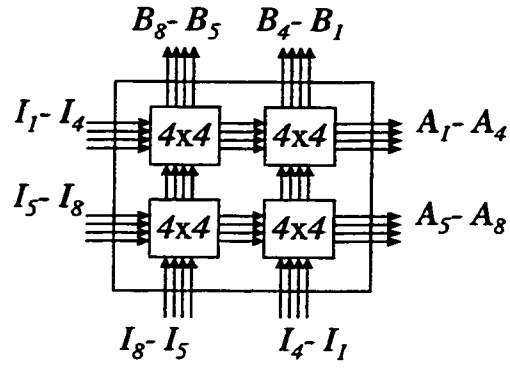


Figure 48- *8x8* Transposer Unit

6.3.2.4 Scalable affine module

The issue of the scalability of reflector and transposer has been discussed in previous sections. We note that in order to design an affine module which performs all selected affine transforms on either four 4x4 blocks or one 8x8 block, special data routing mechanism is required. The module has two modes of operation:

- Operation on *4x4* blocks (*4x4* mode)
- Operation on *8x8* blocks (*8x8* mode)

The configuration of module in each mode is illustrated in Figure 49 and Figure 50, respectively.

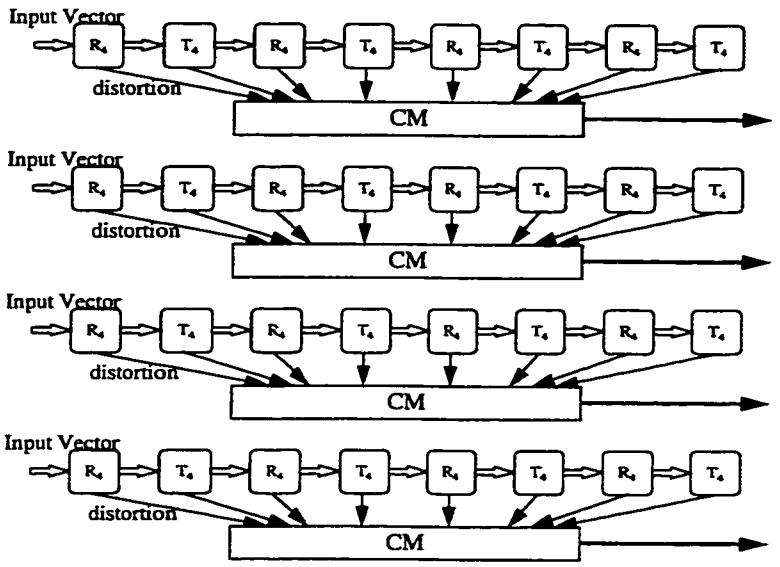


Figure 49- Affine Module for 4x4 blocks

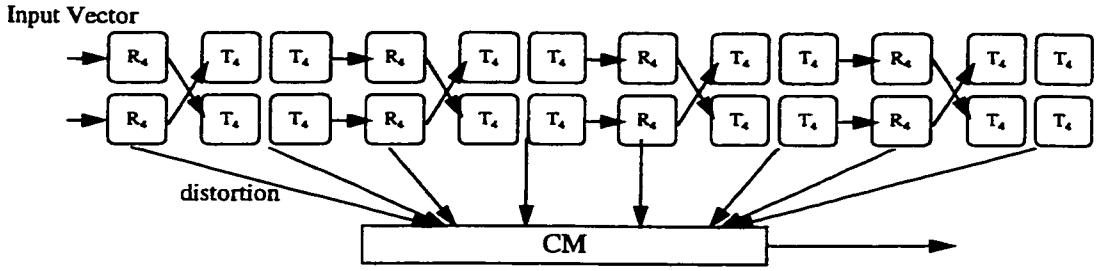


Figure 50- Affine module for 8x8 blocks

The reconfigurable architecture is shown in Figure 51.

We note that this reconfigurable architecture performs data routing in two different modes. In the *4x4* mode, the module processes four *4x4* blocks and is configured as shown in Figure 49. In *8x8* mode, the module processes one *8x8* block and is configured as shown in Figure 50.

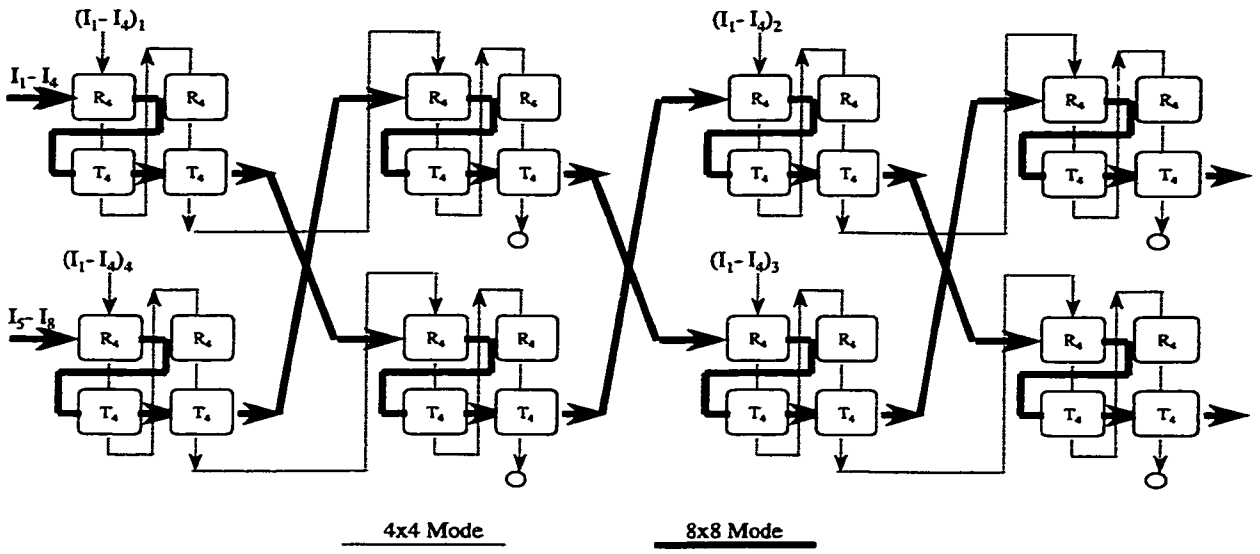


Figure 51- Scalable Affine Module

6.4 Example Algorithms

In this section, we demonstrate the concept of generic processors, real-time execution capability and scalability in Fractal Engine by implementing five examples of compute intensive algorithms. These algorithms include Vector Quantization (VQ)[42], Fractal Block Coding (FBC)[41] and Affine Transform Based Vector Quantization (ATVQ)[25] from spatial domain, Motion Estimation[43] (ME) and Affine Motion Estimation (AME) from temporal domain. These algorithms not only encompass a variety of operations involved in both image and video processing, but also reflect the challenges in visual computing applications from the perspectives of real-time implementation and scalability.

6.4.1 Vector Quantization (VQ)

In VQ[42], a set of representative images is decomposed into *L*-dimensional (*MxM*) vectors. An iterative clustering algorithm such as the LBG algorithm [100] is used to generate a codebook (CB) of size *K*. This codebook is then made available at both the

transmitter and the receiver. In the encoding process, the image to be coded is decomposed into *L*-dimensional vectors. For each input vector V_i (range block), CB is searched using a nearest neighbor rule to find the closest codeword W_j. Compression is achieved by transmitting the label j corresponding to W_j. Reconstruction of images is implemented by using *j* as an address to a table containing the codewords.

The existing high computational complexity in VQ has been an impediment in real-time implementation in many applications. In this section, we demonstrate the real-time implementation of VQ using Fractal Engine. The data flow diagram and the processing architecture of Fractal Engine for VQ execution are illustrated in Figure 52 and Figure 53, respectively.

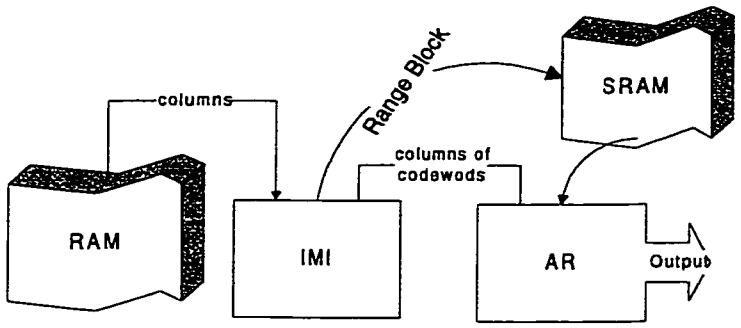


Figure 52 - Data flow in Fractal Engine for VQ Implementation

IMI fetches columns of data including range blocks and codewords from RAM and provides data for AR module. To start with, the first range block is loaded in SRAM. An entire column is loaded in and out of AR cells in each clock cycle. At the end of the first eight clock cycles, the first codeword enters the accumulator module of AR_1. In the second eight clock cycles, the second codeword is loaded into AR_1-ACC, while the partial distortion values are added in AR_1-SUM. The value of distortion between the first

range block and the first codeword is available at the beginning of the third set of eight clock cycles. In the next eight clock cycles, the distortion values are stored in CM_1 for future comparisons. After an initial latency of 24 clock cycles, the utilization factor for AR-1 and CM-1 cells is 100% and at every eight clock cycles, the codewords are compared with the range block.

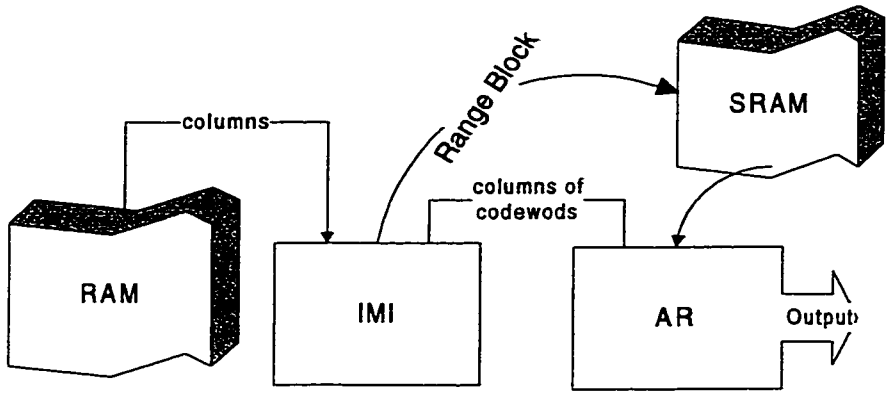
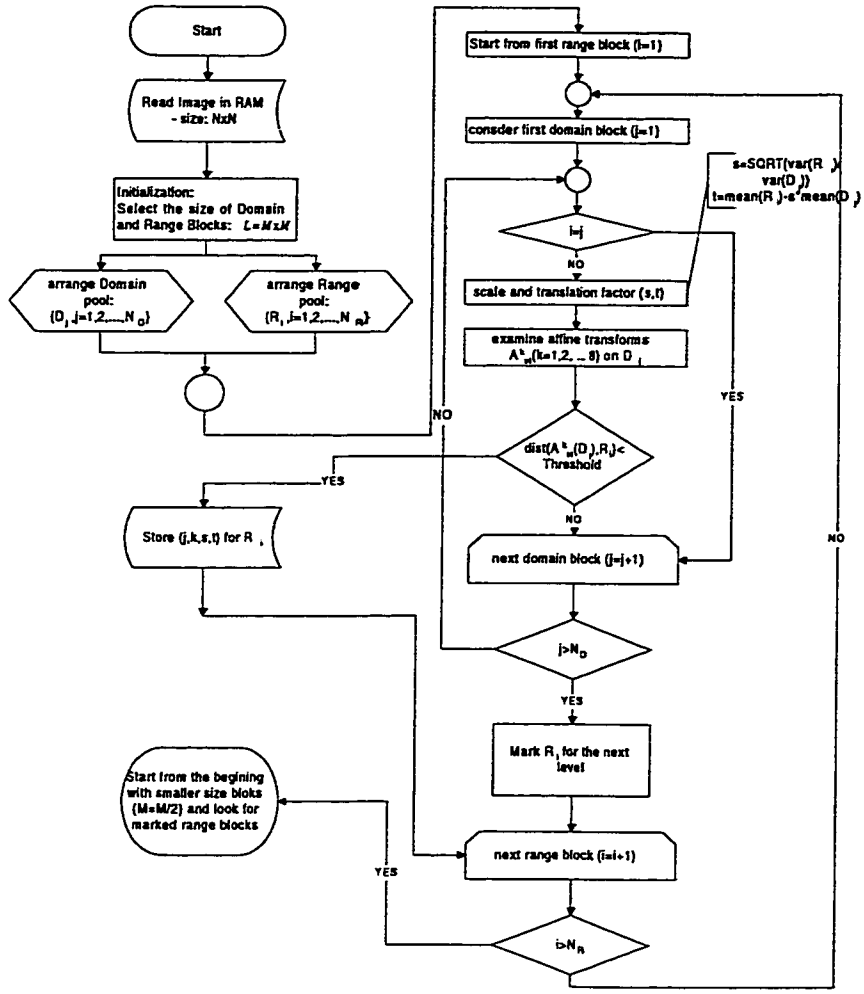


Figure 53 - Processing Section of Fractal Engine for VQ execution

6.4.2 Fractal Block Coding (FBC) Algorithm

Barnsely[36] has proposed an algorithm to compress fractal images with a very high compression ratio (100-10000). This algorithm is based on Iterated Function Systems (IFS). However, real life images are not self-transformable at the level of entire image. A block based fractal image compression method or Fractal Block Coding has been proposed by Jacquin[41] for real life images with a compression ratio ranging from 80 to 200. The proposed algorithm is based on Partial Iterated Function Systems (PIFS).

The sequence of operations in FBC illustrated in a flow chart format is now presented.



1. An original *nxn* monochrome image *f* is partitioned into non-overlapping range blocks, R_i.
2. A pool of domain blocks D_j is made up of all blocks from the original image.
3. For every range block (R_i), the affine contractive transformation (t_i) which minimizes the distortion between R_i and a domain block (D_i) is searched.
4. If the distortion is less than a preset threshold, the best pair (D_i , t_i) is stored.
5. Otherwise, the range block is divided into smaller size range blocks and the search for the best pair (D_i , t_i) is repeated.

6.4.2.1 Implementation of FBC

The modules in Fractal Engine including AFM, SM and ARM are controlled by CU to work in parallel for real-time implementation of FBC. IMI provides data for all units as shown in Figure 54. The procedure consists of two different processes:

1. Mean and variance calculations of all *4x4* and *8x8* blocks.
2. Block matching and affine transformations.

We recall from Section 6, that these two tasks cannot be performed simultaneously on the same frame. Hence, two consecutive frames (f_t and f_{t+1}) are stored in the RAM module. While ARM is calculating the mean and variance values for the range and domain blocks in f_{t+1}, AFM and SCM determine the best candidate domain blocks with appropriate affine transformation for every range block in f_t. The latency is NxN clock cycles, where N is the number of rows or columns of the frame.

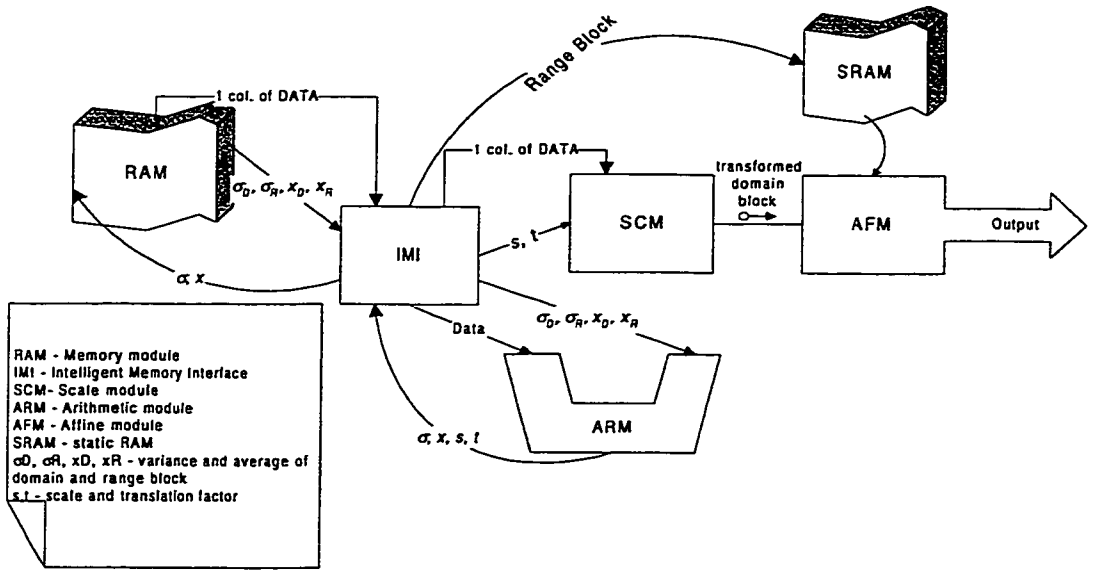


Figure 54- Data flow in Fractal Engine for FBC Implementation.

- **ARM process**

At every clock cycle, one element of the frame is loaded into ARM by IMI. ARM calculates the summation of the elements of a block and their squared values. In every $L=MxM$ clock cycles, where M is the size of the block, the mean and variance of one block is determined and IMI stores the results in the RAM.

- **AFM process**

The modules of processing section in *8x8* mode are shown in Figure 55.

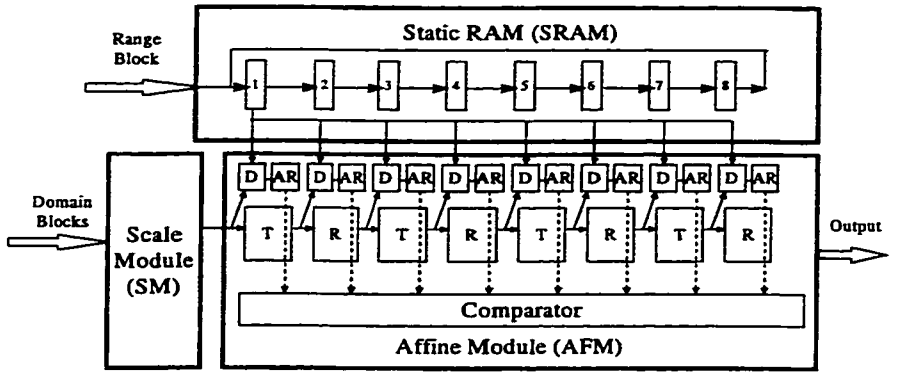


Figure 55- Processing Section in 8x8 mode for Execution.

To start with, the first range block is loaded in SRAM. An entire column is loaded in and out of AFM and SCM in each clock cycle. A column of a domain block (initially the first column) is loaded into SCM. At the end of the first eight clock cycles, the first transformed domain block (B_1) is loaded into R_1 -first reflector module- which is shown in the cell (1,2) of Table 3 and the first affine transformed version of B_1 ($B_1(1)$) enters the accumulator module of AR_1 (1,10). In the second eight clock cycles, the second domain block is loaded into R_1 (2,2) and the first transformed version ($B_2(1)$) into AR_1-ACC (2,10) B_1 moves to T_1 – the first transposer module– (2,3), the second affine transformed of B_1 ($B_1(2)$) enters AR_2-ACC (2,12) and the partial distortion values between range block and $B_1(1)$ enter into AR_1-SUM (2,11). The total value of distortion between the

range block and first affine transformed domain block is available at the beginning of the third set of eight clock cycles and is loaded into the first comparator module (3,26). In the next eight clock cycles, while the total distortion value between the range block and the second domain block is starting to get loaded into CM_1 (4,26), the distortion value between the second affine transformed version of B_1 and the range block is being compared with the previous distortion value (4,27). This procedure continues for all the domain blocks and the process is illustrated in Table 3.

After 72 (*9M*) clock cycles, the best affine transform which generates the least distortion value between the first domain block (B_1) and the range block is available at the output of the comparator module (10,34). We note that the utilization factor after this initial latency is 100%. After processing all the domain blocks, if the minimum distortion is within a pre-specified threshold, the best domain block index and the corresponding affine transform parameters are loaded out of AFM.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	Clock cycle	Block # in R_1	Block # in T_1	Block # in R_2	Block # in T_2	Block # in R_3	Block # in T_3	Block # in R_4	Block # in T_4	AR_1		AR_2		AR_3		AR_4		AR_5	
										ACC	SUM	ACC	SUM	ACC	SUM	ACC	SUM	ACC	SUM
1	0-7	B_1								$B_1(1)$									
2	8-15	B_2	B_1							$B_2(1)$	$B_1(1)$	$B_1(2)$							
3	16-23	B_3	B_2	B_1						$B_3(1)$	$B_2(1)$	$B_2(2)$	$B_1(2)$	$B_1(3)$					
4	24-31	B_4	B_3	B_2	B_1					$B_4(1)$	$B_3(1)$	$B_3(2)$	$B_2(2)$	$B_2(3)$	$B_1(3)$	$B_1(4)$			
5	32-39	B_5	B_4	B_3	B_2	B_1				$B_5(1)$	$B_4(1)$	$B_4(2)$	$B_3(2)$	$B_3(3)$	$B_2(3)$	$B_2(4)$	$B_1(4)$	$B_1(5)$	
6	40-47	B_6	B_5	B_4	B_3	B_2	B_1			$B_6(1)$	$B_5(1)$	$B_5(2)$	$B_4(2)$	$B_4(3)$	$B_3(3)$	$B_3(4)$	$B_2(4)$	$B_2(5)$	$B_1(5)$
7	48-55	B_7	B_6	B_5	B_4	B_3	B_2	B_1		$B_7(1)$	$B_6(1)$	$B_6(2)$	$B_5(2)$	$B_5(3)$	$B_4(3)$	$B_4(4)$	$B_3(4)$	$B_3(5)$	$B_2(5)$
8	56-63	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	$B_8(1)$	$B_7(1)$	$B_7(2)$	$B_6(2)$	$B_6(3)$	$B_5(3)$	$B_5(4)$	$B_4(4)$	$B_4(5)$	$B_3(5)$
9	64-71	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	$B_9(1)$	$B_8(1)$	$B_8(2)$	$B_7(2)$	$B_7(3)$	$B_6(3)$	$B_6(4)$	$B_5(4)$	$B_5(5)$	$B_4(5)$
10	72-79	B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	$B_{10}(1$	$B_9(1)$	$B_9(2)$	$B_8(2)$	$B_8(3)$	$B_7(3)$	$B_7(4)$	$B_6(4)$	$B_6(5)$	$B_5(5)$
11	80-87	B_{11}	B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	$B_{11}(1$	$B_{10}(1$	$B_{10}(2)$	$B_9(2)$	$B_9(3)$	$B_8(3)$	$B_8(4)$	$B_7(4)$	$B_7(5)$	$B_6(5)$

		20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
		AR_6		AR_7		AR_8		Comparator modules								
	Clock cycle	ACC	SUM	ACC	SUM	ACC	SUM	Block# in CM_1	Block# in CM_2	Block# in CM_3	Block# in CM_4	Block# in CM_5	Block# in CM_6	Block# in CM_7	Block# in CM_8	
1	0-7															
2	8-15															
3	16-23							$B_1(1)$								
4	24-31							$B_2(1)$	$B_1(1\text{-}2)$							
5	32-39							$B_3(1)$	$B_2(1\text{-}2)$	$B_1(1\text{-}3)$						
6	40-47	$B_1(6)$						$B_4(1)$	$B_3(1\text{-}2)$	$B_2(1\text{-}3)$	$B_1(1\text{-}4)$					
7	48-55	$B_2(6)$	$B_1(6)$	$B_1(7)$				$B_5(1)$	$B_4(1\text{-}2)$	$B_3(1\text{-}3)$	$B_2(1\text{-}4)$	$B_1(1\text{-}5)$				
8	56-63	$B_3(6)$	$B_2(6)$	$B_2(7)$	$B_1(7)$	$B_1(8)$		$B_6(1)$	$B_5(1\text{-}2)$	$B_4(1\text{-}3)$	$B_3(1\text{-}4)$	$B_2(1\text{-}5)$	$B_1(1\text{-}6)$			
9	64-71	$B_4(6)$	$B_3(6)$	$B_3(7)$	$B_2(7)$	$B_2(8)$	$B_1(8)$	$B_7(1)$	$B_6(1\text{-}2)$	$B_5(1\text{-}3)$	$B_4(1\text{-}4)$	$B_3(1\text{-}5)$	$B_2(1\text{-}6)$	$B_1(1\text{-}7)$		
10	72-79	$B_5(6)$	$B_4(6)$	$B_4(7)$	$B_3(7)$	$B_3(8)$	$B_2(8)$	$B_8(1)$	$B_7(1\text{-}2)$	$B_6(1\text{-}3)$	$B_5(1\text{-}4)$	$B_4(1\text{-}5)$	$B_3(1\text{-}6)$	$B_2(1\text{-}7)$	$B_1(1\text{-}8)$	**B_1**
11	80-87	$B_6(6)$	$B_5(6)$	$B_5(7)$	$B_4(7)$	$B_4(8)$	$B_3(8)$	$B_9(1)$	$B_8(1\text{-}2)$	$B_7(1\text{-}3)$	$B_6(1\text{-}4)$	$B_5(1\text{-}5)$	$B_4(1\text{-}6)$	$B_3(1\text{-}7)$	$B_2(1\text{-}8)$	**B_2**

Table 3 - Cell occupancy for the execution of FBC

6.4.3 Affine Transform Based Vector Quantization

We propose a high performance video compression algorithm[31] which can be ideally mapped onto the Fractal Engine. This algorithm is based on a combined affine transform and vector quantization (ATVQ), where the intra-frame and inter-frame redundancy in a video sequence are exploited through piecewise self-similarity on a block-wise basis within a frame and between frames. In ATVQ, the best match for each vector of the input image is searched among various affine transformed versions of the codewords in addition to the non-transformed codewords (as in standard VQ). Hence, ATVQ can reconstruct more input vectors using a smaller size codebook with a specified distortion compared to conventional vector quantization. In this section, the ATVQ algorithm is mapped onto the Fractal Engine. First, the ATVQ algorithm simulations and its coding performance are discussed followed by the mapping of the algorithm onto the proposed architecture. The timing analysis of the execution of the algorithm is presented in 6.5.1.

6.4.3.1 ATVQ Algorithm

An affine transform based vector quantizer can be defined as a composition of two mappings A_{st}^{k} and Q, where A_{st}^{k} is as introduced in section 5.1, and Q is the conventional vector quantizer which maps R^L into a finite subset Y of R^L.

$$V = A_{st}^{k} \circ Q$$
$$A_{st}^{k}: R^L \rightarrow R^L$$
$$Q: R^L \rightarrow Y$$
$$Y = \{\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_N\}$$

where, $\mathbf{Y}_i$ is an L-dimensional vector.

To start with, a universal codebook is generated which is available at both the transmitter and receiver. The algorithm for codebook generation is detailed in [100]. We note that the codebook generation process is executed only once and is hence executed off-line. The training set for codebook generation includes frames from various video sequences. The steps of the ATVQ algorithm follows:

Step-0. Consider the first frame as the input. (f=1)

Step-1. Partition the input frame into square blocks of size MxM.

Step-2. For each block $\mathbf{X}_i$ select the affine transform A_{st}^k, and the vector $\mathbf{Y}_n$ from the codebook such that:

$$d\left(A_{st}^k(\mathbf{X}_i), \mathbf{Y}_n\right) = \min\, d\left(A_{vw}^l(\mathbf{X}_i), \mathbf{Y}_j\right) \text{ for all possible values of } l, v, w \text{ and } j$$

The algorithm to determine the best affine transform A_{st}^k is now detailed.

For every codeword in the codebook:

a) Calculate the variance and the mean of the input block (*ivar, imean*) and the variance and the mean of the selected codeword (*cvar, cmean*).

b) Assign the scaling factor $s = \sqrt{cvar/ivar}$ and then quantize the value s to the nearest number in the set S.

c) Assign the translation factor $t = cmean - s*imean$. If it is not in the range of set T, the nearest value in the set is selected.

d) Transform the input block to the scaled and translated version.

(i.e. for all x of $\mathbf{X}_i$: $x = s*x + t$)

e) Apply all transforms in the set I to $\mathbf{X_i}$ and calculate the distortion between the transformed block and the codeword.

Determine the least distortion value and store the corresponding values of k, s, t and n.

Step-3. Assign the codeword $[k\ s\ t\ n]$ to $\mathbf{X_i}$.

6.4.3.2 Simulation results

The performance of ATVQ is investigated using 4 test video sequences of 30 frames each (namely, Football, Ping-Pong, Miss America and Salesman). The codebook is generated using the sequences Football, Ping-Pong and Miss America. This codebook is used to code all the test video sequences. Different sets of S and T with varying codeword sizes are employed. The best values for N_s (number of members of S), N_t (number of members of set T) and M (number of rows or columns of each codeword) have been chosen from the results of simulations. The selected values are:

$$N_s = 15 \quad , \quad N_t = 256 \quad , \quad M = 8 \Rightarrow L = 64$$
$$S = \{1/3, 1/2, 2/3, 3/4, 4/5, 5/6, 7/8, 1, 8/7, 6/5, 5/4, 4/3, 3/2, 2, 3\}$$
$$T = \{-128, -127, \cdots, 0, 1, 2, \cdots, 127\}$$

The performance of ATVQ is evaluated using the Rate-Distortion (R-D) criterion, where the distortion is measured using the Peak Signal to Noise Ratio (PSNR) and is defined as:

$$\text{PSNR} = 10 \log_{10}(255 \times 255 / \text{MSE}) \ \text{dB}$$

for 8 bit/pixel (256 gray level) images and MSE is the mean square error between the original image and the reconstructed image. The bit rate for ATVQ is calculated as follows:

$$R_{total} = R_n + R_k + R_s + R_t$$

Where R_n, R_k, R_s and R_t refer to the bitrate for the codeword label, isometric transform index, scaling factor and translation factor respectively and are calculated as follows:

- $R_n = (\log_2 512 / (8 * 8)) = 0.14$ bpp
- $R_k = (\log_2 8 / (8 * 8)) = 0.04$ bpp
- $R_s = (\log_2 16 / (8 * 8)) = 0.06$ bpp
- $R_t = (\log_2 256 / (8 * 8)) = 0.12$ bpp
- $R_{total} = .38$ bpp

The following diagram illustrates the distortion value for all of the test sequences at a bitrate of .38 bpp.

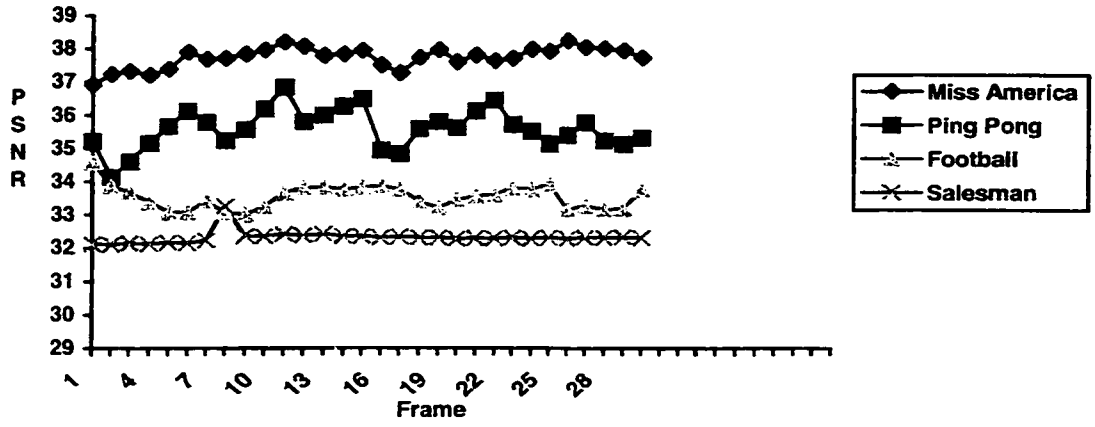


Figure 56 - Performance chart of ATVQ

It can be seen that ATVQ outperforms VQ at the same bitrate.

6.4.3.3 Mapping of ATVQ on Fractal Engine

The architecture of Fractal Engine for ATVQ is similar to that for FBC. The difference lies in the fact that domain blocks in ATVQ are codewords from the codebook while in FBC, the domain blocks are formed from blocks in the same image (frame).

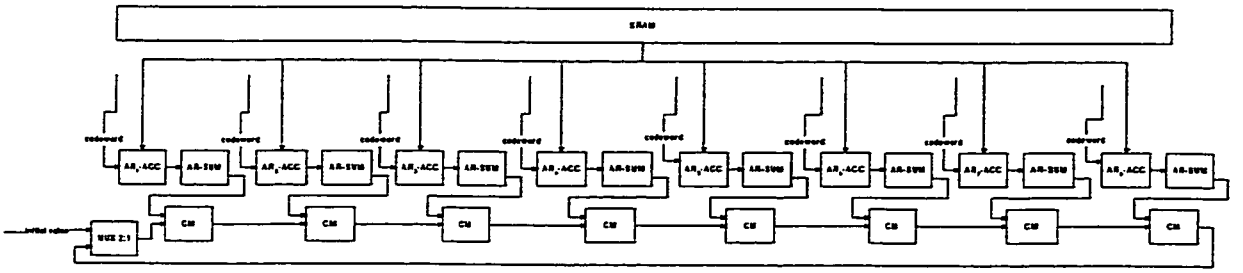


Figure 57 - Processing Section of Fractal Engine for ATVQ execution

All modules in the engine work in parallel and pass data to determine the best match for each range block. The communication between different modules of the Fractal Engine for ATVQ execution is shown in

Figure 58.

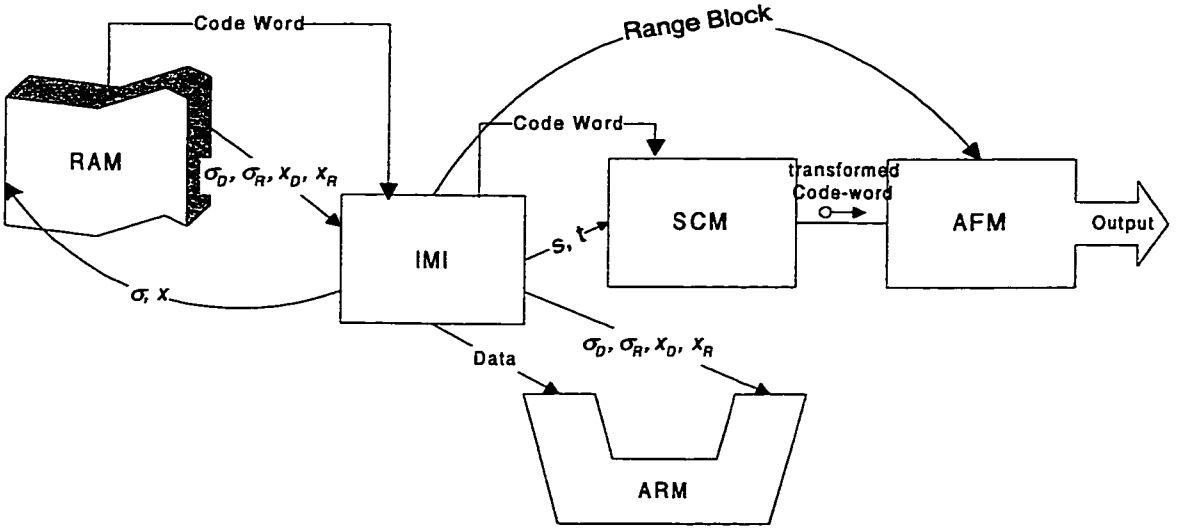


Figure 58 – Data flow diagram of Fractal Engine for ATVQ

In each clock cycle, one column of an *MxM* block from the video sequence enters AFM. After *M* clock cycles, the distortion value between the input block and the stored codeword is calculated and the reflected version of the input block is passed to the next

chain in the cell. In the second *M* clock cycles, the second cell (which is a T-Cell) transposes the input block and calculates its distortion value corresponding to the stored codeword. Meanwhile, the second input block enters the module. Hence, after *9M* clock cycles the distortion values between the transformed codeword and the range block are compared and the best transform is selected. This is fedback for comparison with other distortion values for other codewords. All of the calculated distortion values are compared by **CM** to determine the best affine transform parameters for each codeword in the codebook.

6.4.4 Motion Estimation (ME)

ME[43] is widely used in inter-frame visual media processing particularly in video and image sequences. ME-BMA (block matching algorithm) is typically used in inter-frame motion-compensated (MC) processing. In BMA, motion of a block of pixels (usually *MxM*), within a frame interval is computed. The range of the motion vector is constrained by the search window. BMA assumes that all pixels within the block have uniform motion. The goal is therefore to find the best match between the block in the current frame (range block) and a corresponding block (domain block) in the previous frame within a search window of size $\{(M+2m) \times (M+2m)\}$. In H.261, MPEG-1, and H.263, ME is based on (16x16) luminance blocks.

A variety of techniques have been proposed in literature for ME implementation. They are typically compute intensive and are hence difficult to implement in hardware. Since the core processor of Fractal Engine has been designed by optimizing the implementation for a variety of multimedia operations, ME can be mapped ideally onto the Fractal Engine and implemented in real-time.

We note that motion estimation is essentially a pattern matching process. This process has been fully implemented (in parallel) in the Fractal Engine in the execution of VQ and FBP. The difference here is that instead of codewords or domain blocks from the current frame, the affine module is fed with blocks from the previous frame. Furthermore, affine motion estimation (AME) is also possible in real-time using Fractal Engine. In AME, each range block is also compared with the affine transformed version of the candidate blocks in the previous frame. Hence, better match can be obtained using this process. In the next sections, we demonstrate the real-time execution of Motion Estimation and Affine Motion Estimation using the Fractal Engine.

6.4.4.1 Motion Estimation

The full search BMA-ME is implemented in the Fractal Engine. We recall that in BMA, motion of a block of pixels (MxM), within a frame interval is investigated. The best match between the range block and all possible domain blocks in a search window of size $\{(M+2m)$ x $(M+2m)\}$ is searched by the Fractal Engine.

Full search implies that all blocks formed by any pixel displacement within the search window have to be compared to the range block. In other words, $(2m+1)$x$(2m+1)$ blocks in previous frame are compared to the range block and the closest block is selected. Full search ME is compute intensive and is hence difficult to implement in hardware. Fractal Engine is capable of implementing full search ME-BMA in real-time which is demonstrated for the case of M=8. The structure of Fractal Engine for ME is shown in Figure 59.

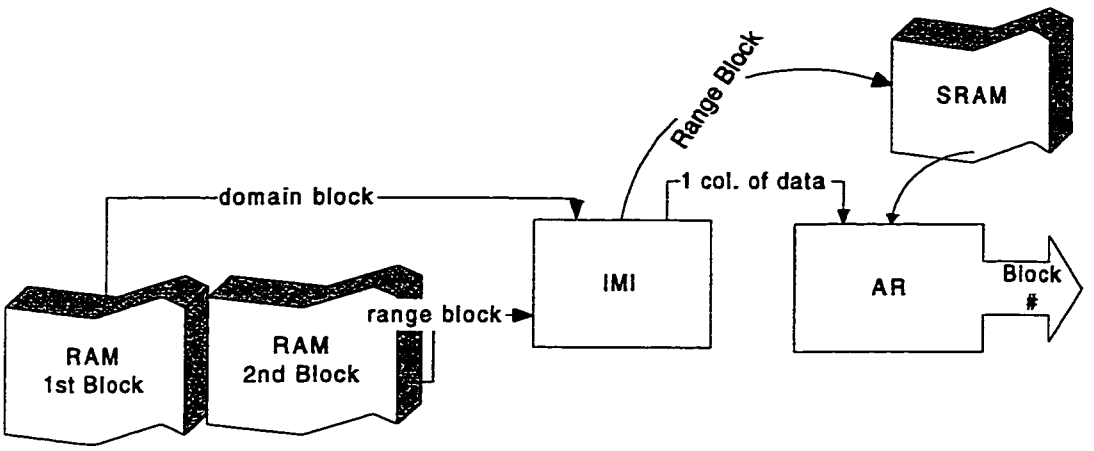


Figure 59 – Data flow diagram of Fractal Engine in ME process

The memory block is divided into two sub-blocks. Each sub-block stores the information of one frame. After the first frame is stored in the first RAM block, motion estimation for the second frame is started. At the same time, the data is stored in the second RAM block. We note that at the end of motion estimation process for the blocks in the second frame (current frame), the contents of the second block of RAM need not be copied to the first RAM block. Instead, the Fractal Engine considers this RAM block as the previous frame data and fills the first RAM block with the new (third frame) information.

6.4.4.2 Affine Motion Estimation

We recall that the main task in data analysis in video applications such as video coding, indexing and compression is motion estimation. The idea is to exploit existing temporal correlation among subsequent frames of a video shot. This kind of correlation exists because in each shot, subsequent frames are taken from one single scenery at different time instances. However, the variances in frames, which are called motion, are due to the movement of objects and various camera operations. Traditional motion estimation

techniques[99],[43] try to model the motion with one-dimensional shift function. This assumption is not valid for complex motion where sophisticated motion functions are required to model the temporal activity. Complex motion functions are not realizable in real time using existing architectures. The affine motion functions capture complex motions and are implementable in real-time using the affine processor of the Fractal Engine. This makes possible analysis of motion in a shot more accurately and hence, outperforms other motion estimation algorithms. In this section, we show the implementation of affine motion function using the Fractal Engine. The basic idea is to find the best match for a range block not only in the domain blocks in previous frame but also in the affine transformed version of those blocks.

The structure of the Fractal Engine for the execution of AME is shown in Figure 60.

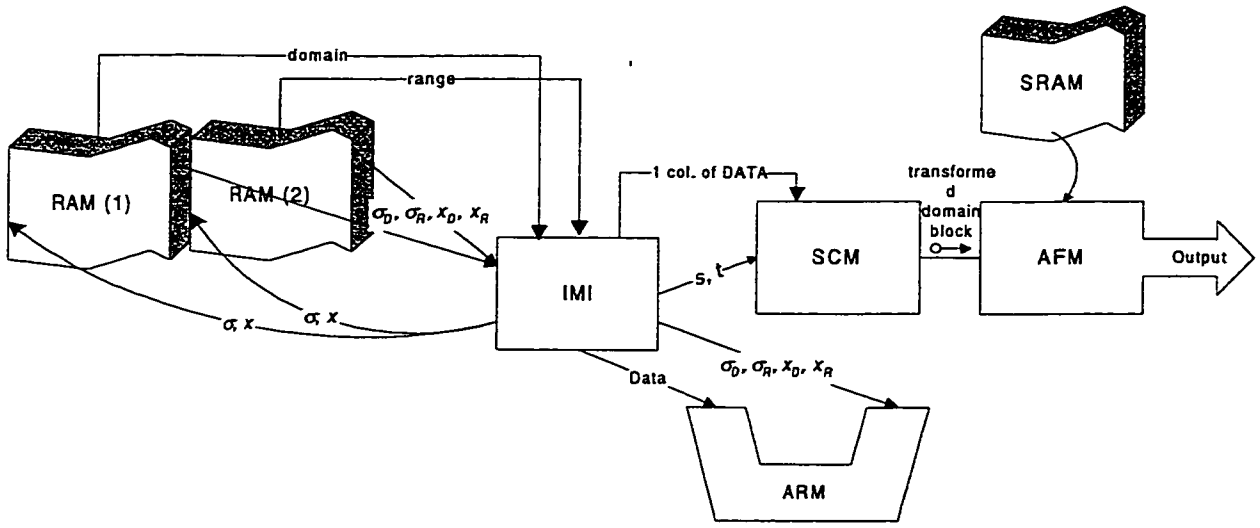


Figure 60 – Data flow diagram of Fractal Engine in AME execution

6.5 VHDL Implementation And Timing Analysis

A behavioral VHDL description of the design has been implemented using the synthesizable part of the VHDL language. The functionality of the design has been tested. After an initial latency of $9M$ clock cycles (where M is the number of rows in a block), the first result becomes available at the output of AFM.

Cell	Ports	Nets	total area*	Max. path delay (ns)
SCM	27	43	184	11.77
AR-ACC	27	56	288	20.72
AR-SUM	29	40	120	11.97
T-cell	26	34	56	0.89
CM	34	48	113	9.17

Table 4- Timing and area analysis of the chip

* area is normalized to the equivalent of a nand2 gate.

The design, has been synthesized (translated and optimized) using BiCMOS .8μ technology. The resulting chip area and speed for the basic modules are shown in Table 4. We note that the area and speed can be further improved by using advanced technology libraries.

The minimum duration of the clock pulse is determined by the maximum of:

- The time taken by the **AR-ACC** to compute the partial distortion value.(20.72 ns)
- The time taken by **AR-SUM** to add the partial values.(11.97 ns)
- The time taken by the **T-cell** to load and transfer the data (0.89 ns)

- The time taken by **SCM** to calculate the translated and scale version of one element of data.(11.77 ns)
- The time taken by CM-cell to compare two distortion values. (9.17 ns)

Hence, the minimum duration of the clock pulse is *20.72* ns and the maximum frequency of operation is $f=1/20.72ns = 48MHz$. We note that by using this specific clock frequency for the Fractal Engine, real-time implementation of the example algorithms are possible. The timing analysis for execution of the algorithms is detailed in next section.

6.5.1 Vector Quantization

The computational complexity of VQ for n range blocks of dimension L for an image of size NxN, and a codebook size K is $O(KLn)$. For example, a 512 x 512 image with vector dimension of $L = 64$ ($8x8$ blocks) encoded using a codebook of size $K = 256$ requires approximately 192 million arithmetic operations.

In the Fractal Engine, after an initial latency, at the end of every eight clock cycles, one codeword is processed. Hence, K=2048 clock cycles are needed to output the codeword label for each input vector. The number of clock cycles required to encode a frame is (N x N) / (M x M) x K = (512 x 512) / (8 x 8) x 2048 = 8388608. Hence, each frame is encoded in:

$$8388608 \times 20.72ns = 0.17 \text{ seconds.}$$

For a video sequence 30 frame/second, Fractal Engine implements the VQ algorithm in real-time. We note that ATVQ algorithm is executed in Fractal Engine in exactly the same time as VQ because all the transformations and comparisons for each codeword is

processed in parallel along with the basic calculation. In other words, Fractal Engine is capable of executing ATVQ algorithm in real-time.

6.5.2 Fractal Block Coding

We now calculate the number of operations involved in FBC algorithm based on general values for the frame size, *NxN*, the block dimension, *L=MxM* and *f=8* geometric transforms in a general purpose processor.

The number of operations involved in FBC depends on the block size and is calculated as follows:

n = (N x N) / L	number of blocks.
n x (n-1) x L	number of additions and multiplications in scaling and translation stage.
n x (n-1) x f x L	number of multiplications in the block matching process.
n x (n-1) x f x 2	number of additions in block matching process.
n x L	number of integer additions and integer multiplications in mean and variance calculation.
n x (n-1) x (f-1)	number of geometric transforms.

In the case of *128x128* pixel frames, the total number of operations are:

8x8 mode: $7.1x10^7$ additions + $3.76x10^7$ multiplications + $2.93x10^7$ integer additions.

4x4 mode: $2.85x10^8$ additions + $1.5x10^8$ multiplications + $1.17x10^8$ integer additions.

In the Fractal Engine, after an initial latency of 72 clock cycles, at each clock cycle one column of *8x8* (or *4x4*) blocks is processed in all of the sub modules of AFM. Hence, 8 (or 4) clock cycles are needed to output the distortion between a range block and all of eight transformed versions of a normalized domain block. The number of clock cycles required to encode a frame is:

8 x (N x N) / (8 x 8) x (N x N) / (8 x 8) 8x8 mode

(4 x (N x N) / (4 x 4) x (N x N) / (4 x 4)) / 4 4x4 mode

If every operation is performed in one clock cycle in the general purpose processor, the number of clock cycles required for encoding one frame will be: $1.38x10^8 + \rho\ 5.52x10^8$, where ρ is the percentage of remaining *4x4* blocks to be coded. For a typical value of ρ=90% and 40MHz clock signal, it takes 15.87 seconds to encode one frame in the sequential processor while the Fractal Engine encodes each frame in .044 seconds. For a video sequence containing frames of size 176x144 pixels with 10 frame/second (QCIF format), Fractal Engine implements the FBC algorithm in real-time.

It is important to note that the scalable feature of Fractal Engine makes possible real-time implementation of larger size and higher frame rate image and video sequences such as CIF, CCIR 601 and HDTV. For example, real-time implementation of FBC for a CCIR601 sequence can be achieved by simply cascading Fractal Engine modules. Fractal Engine is an open architecture and hence can evolve, adapt, and expand to handle a variety of computing tasks and challenges present in other media processing (including visual processing) applications. We note that the performance analysis are based on.8μ BiCMOS technology and available today's technology like .25μ and .18μ will increase the performance both for area and speed resulting in smaller and faster modules.

6.5.3 Motion Estimation

For an image size of NxN, with blocks of MxM and a search window of $(2m+1)x(2m+1)$, $(NxN)/(MxM)x(2m+1)^2$ block comparisons are required to detect motion vectors for all blocks in a frame. In the Fractal Engine, after an initial latency, in the case of M=16, at every sixteen clock cycles, one block is compared to a range block. Hence, for a QCIF (common intermediate format) video sequence with a frame size of 176x144, M=16, m=15 and clock frequency of 48MHz, 30 frames are processed in 0.95 seconds which results in real-time execution.

6.5.4 Affine Motion Estimation

For an image size of NxN, with blocks of MxM and a search window of $(2m+1)x(2m+1)$, $(NxN)/(MxM)x(2m+1)^2x8$ block comparisons are required to detect affine motion vectors for all blocks in a frame. In the case of M=8, after an initial latency of 72 clock cycles, at every eight clock cycles, eight affine transformed versions of one block are compared to a range block. Hence, for a CIF video sequence with a frame size of 352x288, M=8, m=5 and a clock frequency of 48MHz, 30 frames are processed in 0.82 seconds which results in real-time execution of AME by the Fractal Engine.

6.6 Summary

Fractals exploit the high correlation and self-similarities present in visual data within an image or a sequence of images. Fractal Block Processing (FBP) has been proposed as an algorithmic solution to implement the fractal operators for various images. We have presented the design of a Fractal Engine based on an affine video processor, to meet the real-time requirements of FBP. The highly parallel and pipelined architecture of Fractal

Engine enables this processor to perform a variety of compute intensive visual processing applications in real-time. Scalability and modularity issues are addressed in the design of Fractal Engine. To demonstrate the computational power of Fractal Engine vector quantization, fractal block processing, affine transform based vector quantization, motion estimation and affine motion estimation algorithms are mapped onto the Fractal Engine and have been shown to be implementable in real-time. In order to make the Fractal Engine applicable to other applications which involve image and video operations that are not captured by Fractal Processing, augmenting to the Fractal Engine is required. We present examples of augmenting the Fractal Engine in the next chapter.

7 Augmented Fractal Engine

In this chapter, the design of the augmented Fractal Engine is presented. In the design process, we increase the functionality of the Fractal Engine by adding auxiliary modules, which support flexibility of the design, the interface to peripherals and an interpolation filter. Augmented Fractal Engine affords a level of programmability using external control by an external CPU. It also performs all kinds of general linear filtering using the interpolation filter module. First, interpolation in digital domain is detailed and the interpolation filter design is presented. Finally, supporting architectures for programmability features along with peripheral sections are discussed.

7.1 Interpolation in Digital Images

In digital images, the pixels, or picture elements, are limited to lie on a sampling grid, taken to be the integer lattice. The individual pixels are passed through a mapping function such as affine transforms, which generates the new coordinates corresponding to the transform function. The new coordinates, unlike the input sampling points, do not generally coincide with the integer lattice (for example in a rotation transform, if they are not integer multiples of 90 degree rotations). Hence, the new coordinates can take continuous values assigned by the mapping function. The problem is to locate the exact intensity values of the pixel at the integer lattice points. This requires an interpolation stage to fit a continuous surface through data samples, which may then be sampled at arbitrary positions. The accuracy of interpolation has a significant impact on the quality of the output image. Consequently, many interpolation functions have been investigated

to reduce the computational complexity and improve the image quality. Popular interpolation functions include linear, bilinear, nearest neighbor, *etc.* More sophisticated and accurate methods[52] include, cubic spline interpolation and convolution with a sinc function. Although the sinc function is an ideal candidate, it cannot be realized using a finite number of neighboring elements.

In this section, we propose two modifications to classical interpolation methods to maintain the quality along with performance enhancement. We note that whatever the mapping functions and the algorithms of interpolation are, they can be implemented in two different flavors, namely forward and inverse mapping[54], which are now detailed.

7.1.1 Forward Mapping

The forward mapping consists of copying each input pixel onto the output image at positions determined by the X and Y mapping functions. Figure 61 illustrates forward mapping. Each input pixel is passed through the spatial transformation where it is assigned a new output coordinate value. Notice that the input pixels are mapped from a set of integers to a set of real numbers. In the Figure 61, this corresponds to regularly spaced input samples and irregular output distribution.

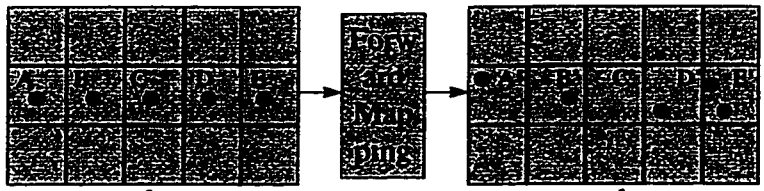


Figure 61- Forward mapping.

In the continuous domain, where pixels may be viewed as points, the mapping is straightforward. However, in discrete domain, pixels are taken to be finite elements defined to lie on a (discrete) integer lattice. It is therefore inappropriate to implement the spatial transformation as a point-to-point mapping. This can result in two types of problems: *holes* and *overlaps*. *Holes*, or patches of undefined pixels, occur when mapping contiguous input samples to sparse positions on the output grid. In Figure 61, C' is a hole since it is bypassed in the input-output mapping. In contrast, *overlaps* occur when consecutive input samples collapse into one output pixel, as depicted in the figure by output pixel E'.

The solution to the point-to-point problem is by using a four-corner mapping paradigm. This considers input pixels as square patches that may be transformed into arbitrary quadrilaterals in the output image.

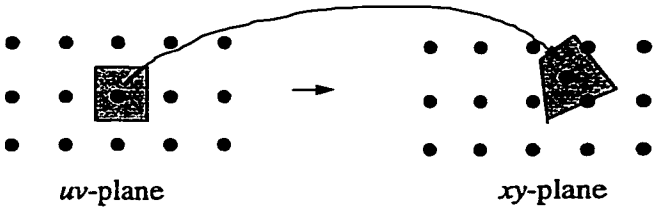


Figure 62- Four corner mapping.

An accumulator array is therefore required to appropriately integrate the input contributions at each output pixel. It is achieved by determining which fragments contribute to each output pixel and then integrating over all contributing fragments. The partial contributions are handled by scaling the input intensity in proportion to the fractional part of the pixel that it spans. Thus, each position in the accumulator array

evaluates $\sum_{i=0}^{N} w_i f_i$, where f_i is the input value, w_i is the weight reflecting its coverage of the output pixel, and N is the total number of deposits into the cell. Using the four-corner mapping solution introduces time consuming intersection tests which precludes real-time implementation for digital rotation.

To overcome this problem, we propose an area mapping algorithm. In this algorithm the point-to-point map is performed. Instead of using a four-corner mapping from the input to the output image, we consider the output pixels as square blocks. The center of each block is located on the coordinates of the mapped point in the output image.

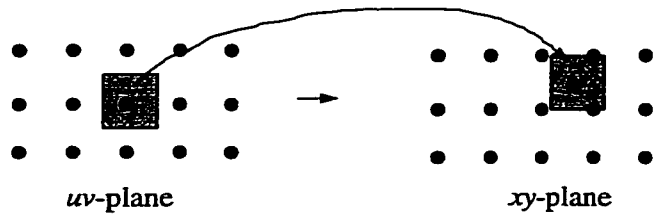


Figure 63- Area mapping.

An accumulator array is then used to evaluate the fractional part of the pixels that it spans.

7.1.2 Inverse Mapping

In inverse mapping, each output coordinate is projected into the input image via $U=X^{-1}$ and $V=Y^{-1}$. The value of the data sample at that point is copied onto the output pixel. This is the most common method since no accumulator array is necessary, and output pixels that lie outside a clipping window need not be evaluated. This method is useful when U and V are readily available (as in the case of most affine transforms) and the input image

can be stored entirely in the memory. Figure 64 illustrates the inverse mapping, with each output pixel mapped back onto the input via the spatial transformation (inverse) mapping function.

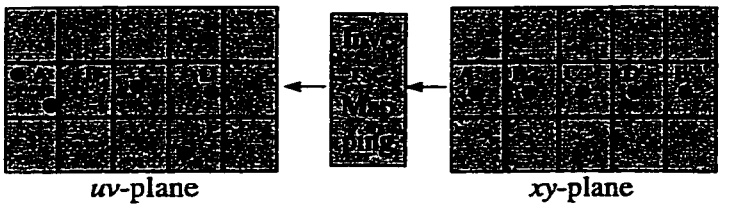


Figure 64- Inverse mapping.

Since the output pixels are projected to the input pixels with real-valued positions, an interpolation stage must be introduced in order to retrieve input values at undefined (non-integral) input positions. Again, area mapping is employed to calculate the intensity of the input pixels at non-integer positions.

7.1.3 Interpolation

Interpolation[51] is the process of determining the values of a function at positions lying between its samples. It achieves this process by fitting a continuous function through the discrete input samples. This permits input values to be evaluated at arbitrary positions in the input, not just those defined at the sample points. Interpolation reconstructs the signal lost in the sampling process by smoothing the data samples with an interpolation function. For equally spaced 1-D data, interpolation can be expressed as

$$f(x) = \sum_{k=0}^{N-1} c_k h(x - x_k)$$

where h is the interpolation kernel weighted by coefficients c_k and applied to N data samples, x_k. Equation (13) formulates interpolation as a convolution operation. Generally, h is a symmetric kernel, i.e. $h(-x)=h(x)$ and c_k coefficients are the data samples. The computation of one interpolated point is illustrated in Figure 65. The interpolating function is centered at x, the location of the point to be interpolated. The value of that point is equal to the sum of the values of the discrete input scaled by the corresponding values of the interpolation kernel. The illustrated interpolation function extends over six points. If x is offset from the nearest point by distance d, where d is between 0 and 1, we sample the kernel at $h(-d)$, $h(-1-d)$, $h(-2-d)$, $h(1-d)$, $h(2-d)$ and $h(3-d)$.

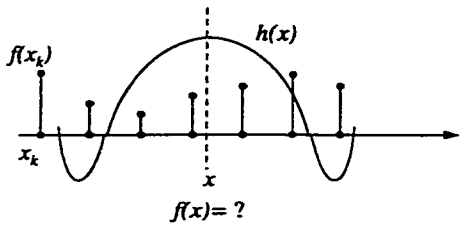


Figure 65- 1-D Interpolation.

Although interpolation has been presented in terms of convolution, it is rarely implemented in this manner. Instead, it is simpler to directly evaluate the corresponding interpolating polynomial at the resampling positions. The discussion of interpolation kernels is necessary due to the comparison between different interpolation techniques.

7.1.4 Interpolation Kernels

The numerical accuracy and computational cost of interpolation algorithms are directly tied to the interpolation kernel[52], [53]. Consequently, interpolation kernels are the

target of design and analysis in the creation and evaluation of interpolation algorithms. They are subject to conditions influencing the tradeoff between accuracy and efficiency.

7.1.4.1 Nearest Neighbor

The simplest interpolation algorithm from a computational standpoint is the nearest neighbor algorithm, where each interpolated output pixel is assigned the value of the nearest sample point in the input image as shown in Figure 66. This technique is expressed by the following interpolating polynomial.

$$f(x) = f(x_k) \qquad \frac{x_{k-1} + x_k}{2} < x \leq \frac{x_k + x_{k+1}}{2}$$

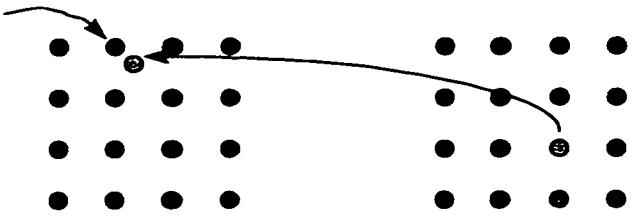


Figure 66- Nearest Neighbor Interpolation.

It can be achieved by convolving the image with a one-pixel width rectangle in the spatial domain. The interpolation kernel for the nearest neighbor algorithm is defined as

$$h(x) = \begin{cases} 1 & 0 \leq |x| < 0.5 \\ 0 & 0.5 \leq |x| \end{cases}$$

Box filter, sample-and-hold function and Fourier window are alternative names for this kernel. This kernel corresponds to multiplication with a sinc function in frequency domain. Due to the prominent side lobes and infinite extent, a sinc function makes a poor low-pass filter. Hence, the nearest neighbor algorithm has a poor frequency domain

response. In this technique, shift errors of up to one-half pixel are possible. For large-scale changes, nearest neighbor interpolation produces images with a blocky appearance. The main advantage of this technique is the simplicity, which makes it possible to implement using a general-purpose processor in real-time.

7.1.4.2 Linear Interpolation

Linear interpolation is a first-degree method that passes a straight line through every two consecutive points of the input signal. Given an interval (x_0, x_1) and function values f_0 and f_1 for the endpoints, the interpolating polynomial is

$$f(x) = f_0 + \frac{x - x_0}{x_1 - x_0}(f_1 - f_0)$$

The corresponding interpolation kernel is

$$h(x) = \begin{cases} 1 - |x| & 0 \le |x| < 1 \\ 0 & 1 \le |x| \end{cases}$$

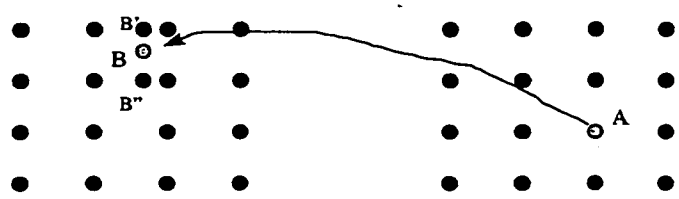


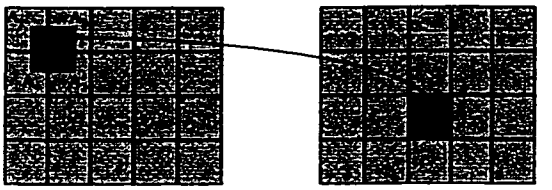
Figure 67- Linear interpolation.

Kernel h is referred to as a triangle filter, tent filter, roof function, Chateau function, or Bartlett window. This interpolation kernel corresponds to a reasonably good low-pass filter in the frequency domain. The side lobes are far less prominent, indicating improved performance in the stop-band. Linear interpolation is widely used for reconstruction since

it produces reasonably good results at moderate cost. 2-D linear interpolation is achieved by separable 1D interpolation as shown in Figure 67. In the first step, two linear interpolations are executed to obtain the intensity value of B' and B" pixels. Then in the next step, another 1D linear interpolation within these two pixels is performed to calculate the intensity value of the desired pixel B.

7.1.4.3 2-D Area Based Interpolation

We recall from section 7.1.1 that a modified algorithm of four-corner mapping will increase the speed while maintaining a similar performance. This algorithm is illustrated for forward mapping in section 7.1.1. In this section, the algorithm is illustrated in Figure 68 for backward mapping.



$$f = \frac{\sum f_i \cdot A_i}{\sum A_i}$$

Figure 68- 2D Area Based Interpolation.

As shown in the figure, the intensity value of the desired input pixel is easily obtained using the fractional part of the pixels that it covers as a weighting function. The justification of this method lies in the fact that the intensity value of each pixel in the

lattice input grid is the average of the illumination received by the sampling device (e.g. scanner).

7.1.5 Experimental Results

In this section, we propose a method to compare the results of different techniques. Since none of interpolation techniques achieves the ideal solution, it is not possible to directly compare the re-produced images. However, we propose to employ the original image as a reference and apply different rotation algorithms as examples of general affine transforms. We then derive the reconstructed image by applying the inverse rotation. The integrity of the reconstructed image compared to the original image is used as a basic for comparison.

The test *256x256* Lena image has 256 gray levels. The employed fidelity criteria are the *Mean Square Error* (MSE) and *Signal to Noise Ratio* (SNR) which are defined in Equation (17) and (18), respectively.

$$MSE = 1/MN \sum_i \sum_j (x_{ij} - y_{ij})^2 \qquad (17)$$

The experimental results for three different interpolation techniques are tabulated in Table 5.

	Nearest Neighbor		Linear		Area Based	
Rotation (deg.)	MSE	SNR(dB)	MSE	SNR(dB)	MSE	SNR (dB)

30	58.26	23.41	53.37	23.64	52.37	23.75
40	64.27	22.99	52.56	23.73	52.17	23.77
45	66.91	22.81	52.72	23.71	52.14	23.76
60	61.52	23.18	53.03	23.69	52.58	23.73

Table 5- Experimental Results.

$$SNR = 10\log\left({}^{1}\!/_{MN} \sum_{i}\sum_{j} (y_{ij})^2 \Big/ MSE \right) \tag{18}$$

It can be seen that the nearest neighbor technique is sensitive to the angle, while the proposed area based and linear interpolation techniques maintain a similar quality for different angles of rotation. The area based algorithm (which can be executed in 8.23 seconds) outperforms the linear interpolation (9.59 seconds) in terms of the speed of operation.

7.1.6 Interpolation Filter Implementation

Interpolation filter belongs to the category of linear filters. We propose a general implementation of these filters which then enables the Fractal Engine not only to perform general interpolation tasks and affine transforms but also all other linear filtering such as DCT and DWT. The general form of a linear filter is expressed in (19).

$$y_i = \sum_{j=0}^{N-1} x_{i-j} . a_j$$

The basic operations in linear filtering are multiplication and accumulation. We now present a pipeline and scalable architecture for accumulation and multiplication.

7.1.6.1 Accumulation

The simple form of accumulation is expressed in (20).

$$Q \leftarrow Q + IN \qquad \text{At every clock cycle} \tag{20}$$

We note that a simple pipeline adder is unsuitable for accumulator, since the adder will stall for multiple clock cycles until the result of previous addition is available before starting the new addition.

Our proposed architecture outputs the result in every clock cycle after an initial latency.

The basic cell of the accumulator is shown in

Figure 69. The cell is a fast, compact and simple adder with an 80 MHz frequency of operation implemented in BiCMOS .8μ technology.

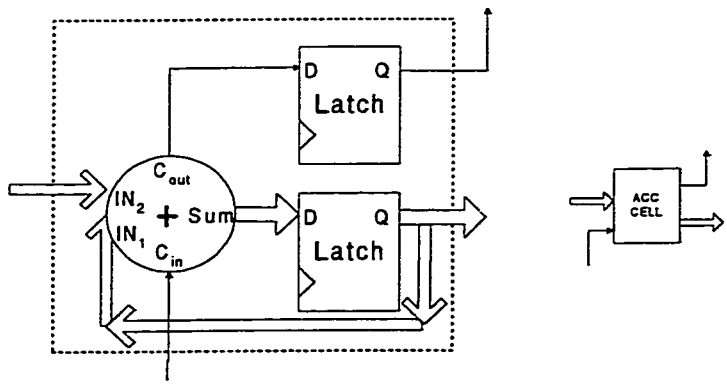


Figure 69 - Basic accumulator cell

The accumulator consists of several ACC-CELL based on the data width. An example of 12 bit accumulator with three level pipeline is shown in Figure 70. We note that there are also load and stop accumulation controls introduced in the design. Load control will set the Q output for a specified value and stop halts the operation of the module.

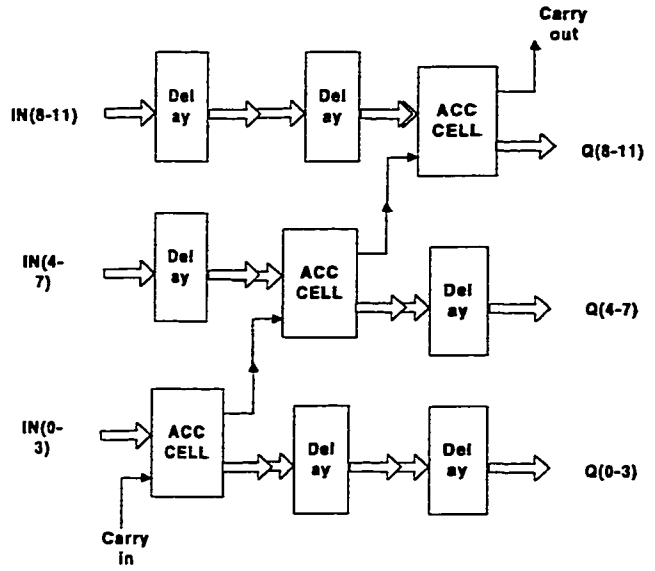


Figure 70 - 12 bit, 3 level-pipelined accumulator

Scalability

The accumulator is modular and scalable. To demonstrate the scalability of the design, we show an example of constructing a 24-bit accumulator with 6 pipeline levels using two 12-bit accumulators and delay modules. The complete design is shown in Figure 71. After the first 3 clock cycles, ACC-1 processes data and the partial result is ready at point C and carry out of the ACC-1 (point B) enters the second unit (ACC-2). At the same time, the input data is output from the delay module D1 at point A and is ready to enter the second

accumulator. After the next 3 clock cycles, the output is ready at points D and E and at every clock cycle, the new output is processed.

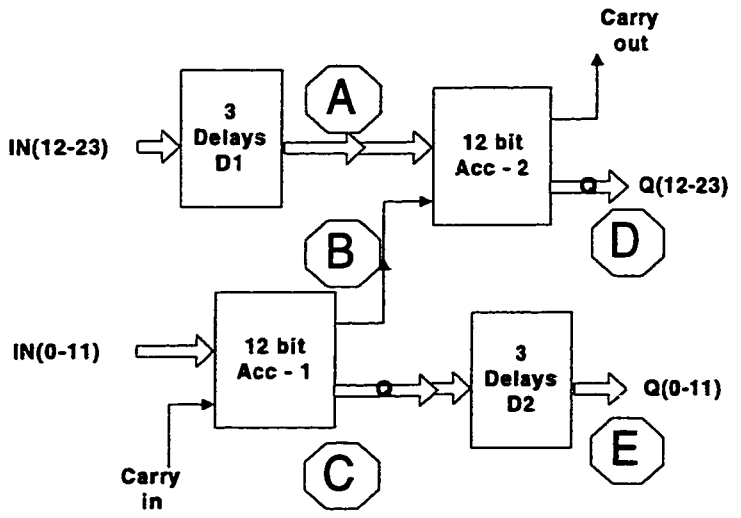


Figure 71 - Scalable accumulator

7.1.6.2 Multiplier

We propose a fast pipelined multiplier based on a 4-bit multiplier. The synthesized 4-bit unsigned multiplier in BiCMOS .8μ has a delay of 15 ns which corresponds to a maximum frequency of 66MHz. This speed of operation is adequate for Fractal Engine and hence, we utilize this cell to implement our multiplier. The block diagram of an 8-bit multiplier is shown in Figure 72.

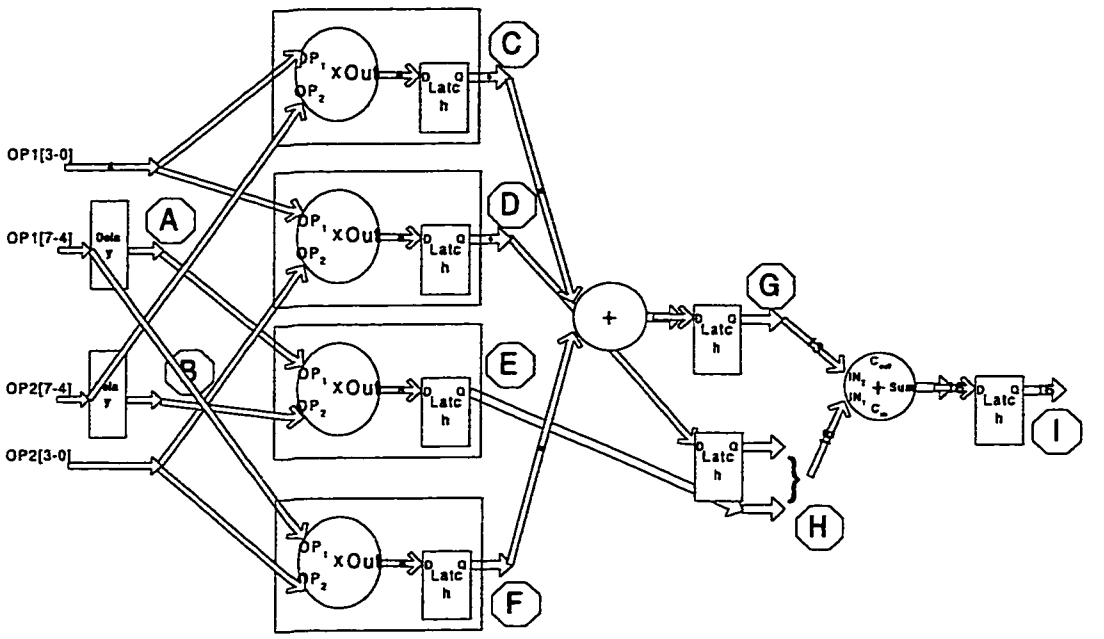


Figure 72 - Block diagram of an 8-bit multiplier

We demonstrate the function of multiplier by an example shown in Figure 73. In the first clock, cycle number A (nibbles a1 and a2) and number B (b1 and b2) are entered into the multiplier. In the next clock cycle, C and D enter the module followed by subsequent operands. The partial results of multiplying the nibbles are shifted accordingly to produce the result as shown in Figure 73.

After the first clock cycle, results a1.b1 at point D (Figure 72), a1.b2 at point C and a2.b1 at point F are ready. In the next clock cycle, a1.b2 plus a2.b1 at point G, and a2.b2 at point E are calculated. At this time, both of a1.b1 and a2.b2 are available in point H and they are appropriately added. In the next clock cycle, the final result is computed and output to point I. After this initial latency of three clock cycles, at every clock cycle the multiplication results are ready and output.

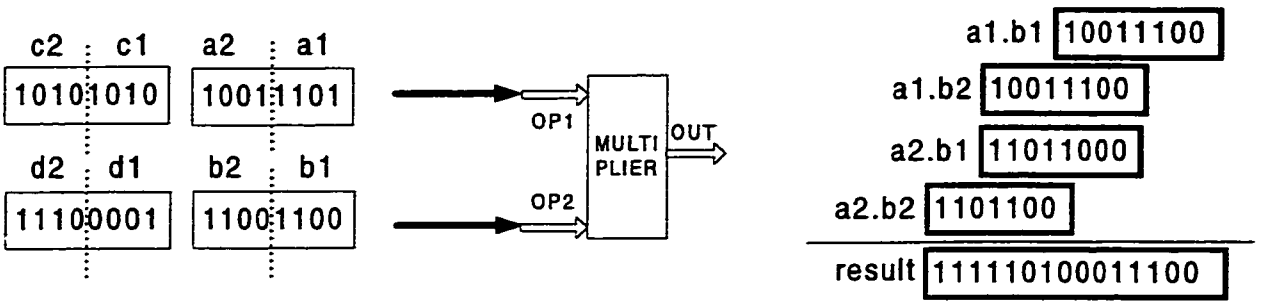


Figure 73 - A multiplication example

7.2 Peripheral Section

Control, communication, storage and interface in the Fractal Engine are implemented in the Peripheral Section. Programmability in the Fractal Engine is acquired by communication between an external CPU with CPU-IF (CPU interface) module. This module is implemented using an SRAM FPGA like an Altera 10K[34] device or Xilinx 4000[35] series.

7.2.1 Random access memory (RAM):

This module stores input data, intermediate results and output data. The description of each item follows:

- Input Data

  The input data essentially consists of the image or the frame to be coded. In the Fractal Engine, two frames are processed simultaneously in different stages. While the second frame enters the ARM for mean and variance calculations, the first frame is loaded into AFM. Hence, two frames are required to be stored in the RAM module.

- Intermediate Results

The mean and variance of every block in the frame are calculated in FBP by ARM. The result is only used to normalize each domain block with respect to a range block and is hence considered as an intermediate result.

- Output Data

  The ultimate output of the system is the bit-stream representing the input frame. For each range block, the index of the closest domain block with the parameters of selected affine transform is stored as output data.

In order to have the maximum speedup, it is important to distribute the data among all modules for concurrent data access. This necessitates parallel and simultaneous accesses to multiple memory elements. Existing architectures for multi-access memory modules fall in two main categories namely multi-port RAM and multi-block RAM.

Multi-port memories with a large number of ports are quite expensive but are however flexible. Parallel access to any combination of memory cells is possible in multi-port memories. In a multi-block RAM architecture, several single port memory modules are employed to store the different blocks of data. Although simultaneous accesses to the memory cells within each block is not feasible, multi-block memory modules have a simpler architecture and occupy less space than multi-port memory modules.

In our design, we present an efficient memory map which fulfills the requirement of all simultaneous memory accesses in the Fractal Engine by using a multi-block memory architecture.

Memory map configuration in the Fractal Engine is as follows:

- One memory block is allocated as a buffer to store the frame data prior to the start of fractal operations. This block has a 14-bit address bus and an 8-bit data bus. As soon as data is loaded into this block, ARM starts the mean and variance calculation.
- The second memory block is allocated to store intermediate results such as the mean and variance of range and domain blocks.
- The major block is allocated to store the image data. Each memory cell and hence, the data bus width is *16x8* bits. Every 16 adjacent pixels are grouped together to form a 128-bit cell of the memory. At every clock cycle, 128 bits of data are read from the memory or written into the memory. In *4x4* mode, every 32 bit segment of the memory cell corresponds to a row of *4x4* blocks in the image and four rows are processed in parallel in the Affine and Scale modules.

7.2.2 Control Unit (CU):

This module essentially consists of finite state machines with different inputs and outputs. The state machines control SRAM, AFM and IMI. It also broadcasts a global signal to all modules to select between 8x8 or 4x4 modes of operation. The implementation of CU and IMI are detailed in Chapter 7.

7.2.3 CPU-IF module

This module communicates with an external CPU to control the execution of Fractal Engine. The CPU can write and read internal registers to and from the FPGA. These registers include:

- Mode flag (write register): This flag is set by the CPU to indicate the 8x8 mode operation for all modules in Fractal Engine. Resetting the flag to 0 implies 4x4 operation of the engine.
- Affine flag (write register): This flag is set to high to indicate that the CM (comparator module) compares the distortion between the affine transformed version of blocks with the range block. When the flag is set to zero, CM compares the range block with the domain blocks only.
- Output register: This register is written by the Fractal Engine. CPU reads this register through CPU_IF module. This register indicates the best candidate for the current range block in process. It shows the domain block number and affine parameters corresponding to the best match for the range block.
- Error register: In addition to the output register, the CPU accesses error register to determine the distortion value between the range block and the best candidate chosen by the Fractal Engine. Based on the value for error threshold, the CPU either accepts the affine parameters or rejects those values.
- Address registers: CPU writes the start and end addresses for a burst transfer, which is executed by the IMI module.

7.2.4 Intelligent memory interface (IMI):

It is important to match the I/O and compute bandwidth in any processor design. Operations must be carefully overlapped, balanced and sequenced to ensure the most efficient use of all the modules in the processor. The IMI ensures that the required data are delivered to the processing modules in parallel and on time. This module acts as a

DMA (Direct Memory Access) device in the Fractal Engine. There are 4 address registers, 2 data registers and 2 counters in IMI for two parallel access to the RAM. CPU sets the start address and end address registers and IMI starts the burst transfer. After a pipeline latency in each clock cycle, data is entered into the affine module of the Fractal Engine. The external selected RAM module is a fast asynchronous static RAM with an access time of 15ns which allows clock frequencies up to 66 MHz. The timing diagram of a typical burst transfer by IMI is shown in Figure 74.

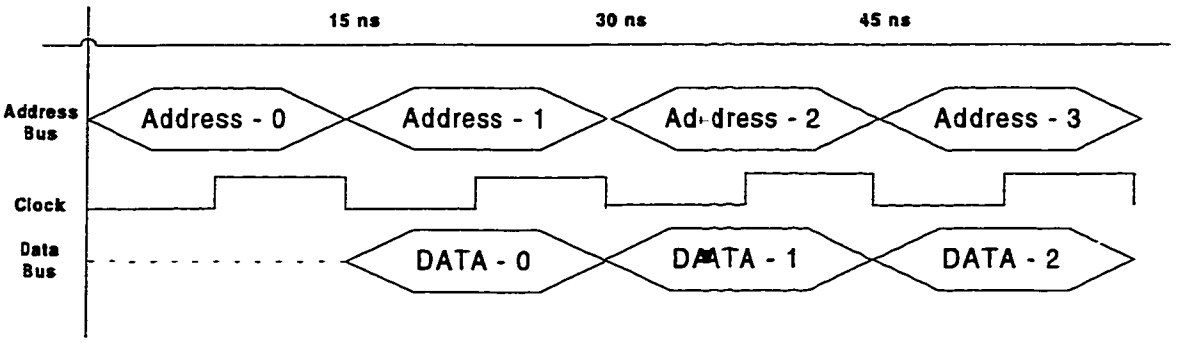


Figure 74 - Burst transfer example for a 66MHz clock

7.3 Summary

Fractal Engine is designed in order to accomplish major multimedia task especially in visual domain in an optimized manner. Several level of enhancements have been applied on the engine. In this section the final stage of completion the Fractal Engine is presented. First, interpolation in digital domain and the interpolation filter is proposed with the new simple and pipelined architectures for accumulators and multipliers. Finally, supporting

architectures for programmability features are added to Fractal Engine chipset. An FPGA implementation for this module is selected due to reprogramability feature of the FPGA. New revisions of FPGA are possible for further control of Fractal Engine.

8 Conclusions

The demands for processing multimedia data in real-time using unified and scalable architectures are ever increasing with the proliferation of multimedia applications. Multimedia processing poses challenges from the perspectives of both hardware and software. In this thesis, we have presented a summary of the various architectural approaches for media processing. Since, visual media represents a significant chunk of the multimedia information, it is crucial to design high performance processors that are reasonably optimized for video processing applications. We have derived the fundamental operations involved in visual processing tasks and designed the generic processing elements to map a majority of these operations. Affine transformations are expected to be increasingly used in many visual processing applications, and hence an affine transform video processing core has been designed. Since Fractal Block Processing encompasses a variety of visual processing operations, we have chosen FBP as the candidate algorithm for the design of the video processor architecture called Fractal Engine. Fractal Engine, which is based on the ATP core, is simple, modular, scalable and is optimized to execute both low level and mid level operations. The individual modules of Fractal Engine have been implemented in VHDL (VHSIC Hardware Description Language). The behavioral description of the Fractal Engine in VHDL has been synthesized towards standard cell ASIC and fast SRAM FPGAs. The function of the Fractal Engine has been demonstrated by mapping popular video processing algorithms such as fractal block coding (FBC), vector quantization and motion estimation. Fractal Engine is capable of processing intra-frame / inter-frame video processing and other media processing applications.

The new design ideas in multimedia architectures such as programmability, scalability and critical dedicated hardware units have been incorporated in the design process of the Fractal Engine.

8.1 Thesis Contributions

The overall contribution of this thesis is in the design of a novel, scalable and optimized visual signal processor termed Fractal Engine. The main contributions are as follows:

8.1.1 Classification of Various Multimedia Operations

- Classification of fundamental operations in visual signal processing (section 2.3).
- Introduction of Fractal Processing as a candidate algorithm to design a visual signal processor called Fractal Engine (section 2.4).

8.1.2 Design Trends in Multimedia Hardware Architectures

- Classification of different design issues including flexibility, processor design, data distribution, memory and granularity (sections 4.1 - 4.5).
- Review of available multimedia processors (section 4.6).
- Analysis of the merits of existing multimedia processors and investigation of the shortcomings (section 4.7).

8.1.3 Hardware / Software Co-design for VLSI Implementation

- Review of enabling VLSI technology including fabrication process and design tools (section 3.2 and 3.3).

- Definition of a new VLSI design methodology employed in the design of Fractal Engine. The methodology is based on a combination of behavioral description of the design (using hardware description languages) and synthesis tools (section 3.3.2).

8.1.4 Affine Transform Processor

- Derivation of generic operations in affine transforms (section 5.2).
- Design of optimized hardware for implementing the generic operations (section 5.3).

8.1.5 Fractal Engine

- Hardware design of Fractal Engine - processing section including Affine module, Scale module and Arithmetic module (section 6.3).
- VLSI Implementation of scalable Affine Transform Processor (section 6.3.2).
- Design of an Intelligent Memory Interface for communication between memory modules and all processing modules (section 7.2.4).
- Augmentation of Fractal Engine by the addition of programmability feature into Control Unit (section 7.2.3).
- Interpolation Filter Implementation in Fractal Engine (section 7.1.6).

8.2 Publications

The contributions of this thesis have been presented and appeared in several refereed international conferences and journals [25], [88], [102]- [109]. An invited paper in the IEEE Transactions on Circuits and Systems for Video Technology[110] is a highlight of the contributions of this thesis.

9 Future Work

Fractal Engine is capable of implementing a variety of visual media processing applications. It is an open architecture and is therefore extendable to implement future multimedia algorithms. Several challenges from the point of view of mapping new algorithms as well as augmenting the Fractal Engine constitute the future work. We now present a sampling of the promising directions of future implementations in the Fractal Engine. In the first section, we present some of the existing multimedia algorithms, which are implementable in the Fractal Engine. In the second session, we propose new algorithms based on affine transforms and fractal processing. These algorithms can be ideally mapped onto the Fractal Engine.

9.1 Multimedia Algorithms

- Scene Cut Detection – optical flow
  - The inputs to video processing algorithms are frames belonging to a shot. However, a video sequence typically contains several shots. Scene cut detection algorithms partition video into shots by detecting the shot boundaries (scene cuts) using optical flow techniques (similar to motion estimation) which can be directly mapped onto the Fractal Engine.
- Discrete Cosine Transform (DCT)
  - DCT is widely used in image and video applications because it offers the closest performance to the computationally expensive KL transform. DCT

calculation is based on array multiplication and accumulation which can be performed in parallel using the Fractal Engine.

- Discrete Wavelet Transform (DWT)
  - Recently, wavelet theory has emerged as a powerful technique for non-stationary signal analysis. The implementation of DWT is very similar to sub-band coding. Wavelets offer a variety of useful features in image and signal processing. DWT calculation is based on array multiplication and accumulation, which can be performed in parallel using the Fractal Engine.
- MPEG-4 and MPEG-7 standards
  - Upcoming MPEG-4 and future MPEG-7 standards are expected to involve a variety of video signal processing algorithms including content based coding, sprite coding, mesh and phase animation coding, affine transformations and indexing. This would require a generic open architecture for video signal processing implementation such as Fractal Engine.

9.2 New Affine Algorithms

- Affine Motion Estimation (AME)
  - We recall from section 6.4.4.2 that complex motion estimation requires implementation of sophisticated affine motion functions. AME algorithms are implementable in real-time using the affine processor of the Fractal Engine.
- Fractal Video Compression

- Fractal techniques are typically used in image processing applications. We note that the basic element in fractal video algorithms is affine transform and hence, the Fractal Engine is a perfect choice to execute these algorithms.

- Camera operation detection using affine transforms

  - There are two sources for pixel displacement within the frames of any video shot namely object motions and camera operations. Camera operations like panning and zooming introduce sophisticated motion patterns in image sequences. These patterns can be captured precisely using affine operations and hence can be implemented in the Fractal Engine.

10 References

[1] B. Tarim and M. Ismail, "Enhanced anolaog yields cost-effective systems-on-chip", IEEE Circuits & Device Magazine. Vol. 15 no. 2 pp 12-22, 1999.

[2] H. Sasaki, "Future trend of VLSI technology and business", International Symposium on Plasma Process-Induced Damage, P2ID, Proceedings 1998, pp 1-6.

[3] J. Wilson et al.,"Challenges and Trends in Processor Design", Computer Vol. 31 no. 1 Jan. 1998.

[4] I. Watson, "Internet, Intranet, Extranet: managing the information bazaar", ASLIB Proceedings. Vol. 51 no. 4 pp 109-114, Apr. 1999.

[5] E. Ayanoglu, "Wireless Broadband and ATM systems", Computer Networks-The International Journal of Computer & Telecommunications Networking. Vol. 31 no. 4 pp 395-409, Feb. 1999.

[6] V. Bhaskaran and K. Konstantinides, "Image and Video Compression standards", Kluwer Academic Publishers, 1996.

[7] G. K. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, Vol. 34, No. 4, pp. 30-45, April 1991.

[8] MPEG (Moving Pictures Expert Group), Final text for ISO/IEC 11172, *Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbits/s*, ISO/IEC, 1993.

[9] M. O'Docherty and C. Daskalakis, "Multimedia information systems: the management and semantic retrieval of all electronic data types," *The Computer Journal*, Vol. 34, No. 3, pp. 225-238, 1991.

[10] Digital Libraries, Special issue of *Communications of the ACM*, Vol. 38, No. 4, April 1995.

[11] M. Ehlers, G. Edwards and Y. Bedard, "Integration of remote sensing with geographic information systems: a necessary evolution," *Photogrammetric Engineering and Remote Sensing*, Vol. 55, No. 11, pp. 1619-1627, 1989.

[12] C. Tomlin, *Geographic Information Systems and Cartographic Modeling*, Prentice Hall, Englewood Cliffs, NJ., 1990.

[13] Horace Newcomb, "Encyclopedia of television", ISBN: 1884964249, Chicago : Fitzroy Dearborn Pub., 1997

[14] D. L. Gall, "MPEG: A Video compression Standard for Multimedia Applications", Communications of the ACM, Vol. 34, No. 4, April 1991, pp. 59-63.

[15] RE. Anderson, *etat.* "Intergrationg the MPEG-2 sybsystem for digital television", IBM Journal of Reasearch & Development. Vol. 42 no. 6 pp 795-805, Nov. 1998.

[16] D. Anastassiou, "Current status of the MPEG-4 standardization effort," SPIE/VCIP, vol. 2308, pp. 16-24, Chicago, IL, Sep. 1994.

[17] Y. Q. Zhang, "Very low bitrate video coding standards," *Proc. of SPIE: Visual Communications and Image Processing*, pp. 1016-1023, May 1995.

[18] A. Tabatabai, M. Mills and M. L. Liou, "A review of CCITT *p*x64 kbps video coding and related standards", Intl. Electronic Imaging Exposition and Conf., pp. 58-61, Oct. 1990.

[19] ITU-T Recommendation H.263. "Video Coding for Low Bitrate Communication", October 1995.

[20] "MPEG-7 FAQ", http://drogo.cselt.stet.it/mpeg/faq/faq_mpeg-7.htm.

[21] K. Aizawa and T. S. Huang, "Model-Based Image Coding: Advanced Video Coding Techniques for Very Low Bit-Rate Applications", Proceedings of the IEEE, Vol. 83, No. 2, February 1995, pp. 259-271.

[22] H. Liu *et al.* "Trifocal Motion Modeling for Object-Based Video Compression and Manipulation", IEEE Transactions on Circuits & Systems for Video Technology. Vol. 8 no. 5 pp667-685, Sep. 1998.

[23] M. Strintziz and S. Malassiotis, "Object-based coding of stereoscopic and 3D image sequence", IEEE Signal Processing Magazine. Vol. 16 no. 3 pp 14-28 May 1999

[24] A. Saflekos, *et al.*, "Coding of 3D Moving Medical Data Using a 3D Warping Technique", Signal Processing Vol. 55 no. 2 pp 247-252, Dec. 1996.

[25] O. Fatemi and S. Panchanathan, "VLSI Chip-Set for Affine Based Video Compression", SPIE Proceedings Vol. 2668 pp. 233-242, Feb. 96.

[26] Y. Hara and J. Yoshida " U.S., Japan IC makers race to media processors" Electronic Engineering Times, Issue 1010 June 1998.

[27] Michael Kagan, "P55C Micro-Architecture: The First Implementation of the MMX Technology", Proceedings IEEE Hot Chips 8, Stanford CA, Aug. 1996. http://infopad.eecs.berkeley.edu/HotChips8/5.2/.

[28] K. Guttag, R. J. Gove and J. R. Van Aken, "A Single-Chip Multiprocessor For Multimedia: The MVP", IEEE Computer Graphics and Applications vol. 12 no. 6, Nov. 1992.

[29] Gerrit Slavenburg, Selliah Rathnam, and Henk Dijkstra, "The TriMedia TM-1 PCI VLIW Media Processor", Proceedings IEEE Hot Chips 8, Stanford CA, Aug. 1996. http://infopad.eecs.berkeley.edu/HotChips8/6.1/.

[30] http://www.c-cube.com/pressrls/press3-3-96.html & http://www.c-cube.com/prdctlst/products.html#996933

[31] W. Wolf, "Modern VLSI Design", Prentice-Hall, 1994.

[32] Z. Navabi, "VHDL: analysis and modeling of digital systems", New York, McGraw-Hill. 1993

[33] Richard C. Seals and G. F. Whapshott, "Programmable Logic: PLDs and PGAs", McGraw Hill Text; ISBN: 0070572607, Apr. 1997.

[34] "ALTERA Data Book", June 1998.

[35] "The Programmable Logic Data Book", Xilinx 1998.

[36] M. F. Barnsely and S. Demko, "Iterated function systems and the global construction of fractals", Proc. Roy. Soc. London, vol. A399, pp. 243-275, 1985.

[37] R. C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison Wesley, 1992.

[38] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, 1989.

[39] J. Beaumont, "Image data compression using fractal techniques", BT Technology Journal. Vol. 9 no. 4 Oct 1991 pp 93-109.

[40] "CMC Fabrication Technologies", http://www.cmc.ca/Fabrication/faboutlook.html.

[41] A. E. Jacquin, " Fractal Image Coding: A Review", Proceeding of the IEEE; vol. 81, No. 10, pp. 1451-1465, October 1993.

[42] F. Idris and S. Panchanathan, "Image Sequence Coding Using Frame Adaptive Vector Quantization", Visual Communications and Image Processing '93, Vol. 2094, pp. 941-952, November 1993.

[43] R. Srinivasan and K. Rao, "Predictive coding based on motion estimation," *IEEE Trans. on Communications*, Vol. COM-33, pp. 888-896, Aug 1985.

[44] Jähne, Bernd, "Digital image processing: concepts, algorithms, and scientific applications", Berlin ; New York : Springer-Verlag, c1991.

[45] D. Westerkamp and H. Peters, "Comparison between progressive and interlaced scanning for a future HDTV system with digital rate reduction," in *Signal Processing of HDTV*, L. Chiariglione, Ed., pp. 15-23, Amsterdam, 1988.

[46] A. M. Tekalp, *Digital Video Processing*, Prentice Hall, New Jersey, 1995.

[47] H. Li, K. Liu,and S. Lo, "Fractal modeling and segmentation for the enhancement of microcalcifications in digital mammograms", IEEE Transactions on Medical Imaging. Vol. 16 no. 6 pp 785-798, Dec. 1997.

[48] M. Makamura, M. Masuda and K. Shinohara, "Multiresolutional image analysis of wood and other materials", Jounal of Wood Science. Vol. 45 no. 1 pp 10-18, 1999.

[49] A. Conci and C. Proenca, "A Fractal image Analysis system for fabric inspection based on a box-counding method", Computer Networks & ISDN systems. Vol. 30 no. 20-21 pp 1887-1895, Nov. 1998.

[50] T. Sato, M. Matsuoka and H. Takayasu, "Fractal Image Analysis of Natural Scenes and Medical Images", Fractals Vol. 4 no. 4 pp 463-468 Dec. 1996

[51] V. Rasche, *et al.*, "Resampling of data between arbitrary grids using convolution interpolation", IEEE Transactions on Medical Imaging. Vol. 18 no. 5 pp 385-392, May 1999.

[52] C. Banert and H. Prautzsch, "Quadric splines", Computer Aided Geometric Design. Vol. 16 no. 6, pp 497-515, Jul. 1999

[53] R. Hanssen and R. Balmer, "Evaluation of interpolation kernels for SAR interferometry", IEEE Transactions on Geoscience & Remote Sensing. Vol. 37 no. 1 pp 318-321 Jan. 1999.

[54] W. Lawton, "A Fast Algorithm to Map Functions Forward", Multidimensional Systems & Signal Processing. Vol. 8 no. 1-2 pp 219-227 Jan. 1997.

[55] N. Chen and W. Zhu, "Bud-Sequence Conjecture on M Fractal Image and M-J conjecture between C and Z planes", Couputers & Graphics. Vol. 22 no. 4 pp 537-546, Jul.-Aug. 1998.

[56] C. Kim, R. Kim and S. Lee, "A Fractal Vector Quantizer for Image Coding", IEEE Transactions of Image Processing. Vol. 7 no. 11 pp 1598-1602, Nov. 1998.

[57] P. Palazzari, M. Coli and G. Bulli, "Massively parallel processing approach to fractal image compression with near-optimal coefficient quantization", Jonal of Systems Architecture. Vol. 45 no. 10 pp765-779, Apr. 1999

[58] "Intel to Add 100-MHz Bus to Pentium Pro", Computer. Vol. 30 no. 3, Mar. 1997

[59] R. Myrvaagnes "POWERPC Architecture Gains Vector Processing Capability", Electronic Products Magazine. Vol. 41 no. 3 Aug. 1998

[60] D. Bursky, "Enhanced POWERPC Architecture Delivers Vector Processing, Wider Internal Bus", Electronic Design Vol. 46 no 18 Aug. 1998

[61] J. Krause, "PowerPC G(3) aims for 400(+) MHz in 98", Byte Vol 23 no 4 Apr. 1998

[62] N. Magotra, *et al.*, "Digital image processing applied to imaging interferometric lithography", Proceedings of the 1998 32nd Asilomar Conference on Signals, Systems & Computers. Vol. 2 pp 989-993, 1998.

[63] J. Procter and N. Rothwell, "Silver: An Integrated Composition System For Vlsi Design", IEE Colloquium (Digest). Publ by IEE, London, Engl p 7. n 1986/68. 1-7.

[64] R. Raud, "Language environment for ASIC design", Microprocessing & Microprogramming. Vol. 24 no. 1, pp 219-226, Aug 1988.

[65] M. Dolle, et al., "A 32-B RISC/DSP Microprocessor with Reduced Complexity", IEEE Journal of Solid-State Circuits. Vol. 32 no. 7 pp 1056-1066, Jul. 1997.

[66] J. Gonzalez and A. Gonzalez, "Data value speculation in superscalar proceesors", Microprocessors and Microsystems. Vol. 22 no. 6 pp 293-301, Nov. 1998

[67] N. Lu and C. Chung, "Parallelism exploitation in superscalar multiprocessing", IEE Proceedings-E Computers & Digital Techniques. Vol. 145 no. 4 pp 225-264, Jul. 1998.

[68] A. Hutton, "The Embedded Superscalar Revolution", Microelectronics Journal. Vol. 29 no. 8 pp 547-551, Aug. 1998.

[69] A. De Gloria, "Microprocessor design for Embedded System", Journal of Systems Architecture. Vol. 45 no. 12-13 pp 1139-1149, Jan. 1999.

[70] C. Basoglu and Y. Kim, "A new couse on superscalar and VLIW computer architectures for real-time image and video computing", IEEE Transactions on Education. Vol. 41 no. 4, Nov. 1998

[71] A. Desouza, E. Fernandes and A. Wolfe, "On the Balance of VLIW Architecture", Vol. 43 no. 1 pp 15-22 Mar. 1997.

[72] R. Alderman, "Serial Dataflow Architectures", Electronic Design. Vol. 46 no. 21 Sep. 1998

[73] G. Tziritas and C. Labit, "Motion analysis for image sequence coding", Amsterdam, Netherlands: Elsevier Science, B.V. 1994.

[74] R. Clarke, "Image and video compression: A survey", International Journal of Imaging Systems & Technology. Vol. 10 no. 1 pp 20-32, 1999

[75] K. A. Birney and T. R. Fischer, "On the modelling of DCT and subband image data for compression," *IEEE Trans. on Image Processing,* Vol. 4, pp. 186-193, Feb. 1995.

[76] K. Guttag, R. J. Gove and J. R. Van Aken, "A Single-Chip Multiprocessor For Multimedia: The MVP", IEEE Computer Graphics and Applications vol. 12 no. 6, Nov. 1992.

[77] Steve Purcell, "The Impact of Mpact 2", IEEE Signal Processing Magazine, Vol. 15 No. 2 pp. 102-107, March 1998.

[78] K. Suzuki, *et al.* "V830R/AV – Embedded Multimedia Superscalar Risc Prcessor", IEEE Micro. Vol. 18 no. 2 pp36-47 Mar.-Apr. 1998.

[79] H. Terada, S. Miyata and M. Iwata, "DDMP's: Self-timed super-pipelined data-driven multimedia processors", Proceedings of the IEEE. Vol. 87 no. 2 pp 282-296, Feb. 1999.

[80] http://www.han.com/~woobin/papers/EI94/main.html

[81] http://www.c-cube.com:80/techsprt/faq/qavrisc.html

[82] http://www.lsilogic.com/products/unit5_6d.html

[83] http://www.raleigh.ibm.com/vip/vipinfo.html

[84] www.8x8.com/8x8/products/vcp.html

[85] http://array.com/product.html

[86] Angel L. Decegama, "Parallel Processing Architectures and VLSI Hardware", Prentice Hall, 1989.

[87] Brian Hayes, "A Computer with Its Head Cut Off", American Scientist, Vol. 83 pp. 126-130, Mar.-Apr. 1995.

[88] O. Fatemi and S. Panchanathan, "FPGA Implementation Of A Matrix Transposer", June 1994, Proc. Canadian Workshop on Field-Programmable Devices pp. 4.4.1-4.4.7, Kingston, Canada.

[89] "LSI Logic Consumer Products", http://www.lsilogic.com/products/unit5_6z.html.

[90] S. Bose, "A single chip multi-standard video codec", Proceedings IEEE Hot Chips V, Stanford CA, Aug. 1993.

[91] "8x8's Video Communication Processor", http://www.8x8.com/docs/chips/vcp.html.

[92] Phil Bernosky and Scott Tandy, "Bringing Workstation Graphics Performance to a Desktop Near You: ViRGE/VX", Proceedings IEEE Hot Chips 8, Stanford CA, Aug. 1996. http://infopad.eecs.berkeley.edu/HotChips8/9.2/.

[93] "MediaGX Architectural System Overview", http://www.cyrix.com/process/prodinfo/mediagx/gxovervw.htm.

[94] M. Antonini, M. Barlaud, P. Mathieu and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. on Image Processing*, Vol. 1, No. 2, April 1992.

[95] C. Caffario, C. Guaragnella, F. Bellifemine, A. Chimienti and R. Picco, "Motion compensation and multiresolution coding," *Signal Processing : Image Communication*, Vol. 6, No. 2, pp. 123-142, May 1994.

[96] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, Philadelphia, 1992.

[97] R. A. Devore, B. Jawerth and B. J. Lucier, "Image compression through wavelet coding," *IEEE Trans. on Information Theory*, Vol. 38, No. 12, pp. 719-746, March 1992.

[98] gemoetric transforms

[99] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. on Communications*, Vol. COM-29, pp. 1799-1806, Dec 1981.

[100] Y. Linde, A. Buzo and R. Gray, "An algorithm for vector quantizer design", *IEEE Trans. on Communications*, pp. 84-95, Jan 1980.

[101] C. E. Shanon, "Coding theorems for a discrete source with a fidelity criterion," *IRE National Convention Record*, Part 4, pp. 142-163, 1959.

[102] O. Fatemi and S. Panchanathan, "Real-Time VLSI Architecture for Video Compression", Proc. Of the Canadian Conference on Electrical and Computer Engineering, Vol 1, pp 128-131, September 95, Montreal, Canada.

[103] O. Fatemi, S. Zhang and S. Panchanathan, "Optical Flow Based Model for Scene Cut Detection", Proc. Of the Canadian Conference on Electrical and Computer Engineering 96, pp 470-473, Calgary, Canada.

[104] O. Fatemi, F. Idris and S. Panchanathan, "FPGA Implementation Of The LRU Algorithm For Video Compression", IEEE Transactions on Consumer Electronics Vol 40, No 3 pp. 337-344, August 1994.

[105] O. Fatemi and S. Panchanathan, "VLSI Architecture of a Scalable Matrix Transposer", Proceedings of Eight International Conference on Innovative Systems in Silicon, ISIS' 96, pp 382-391, Austin, Texas, USA.

[106] O. Fatemi and S. Panchanathan, "Fractal Engine", SPIE' 97 (Multimedia Hardware Architectures), SPIE Vol. 3021 pp 88-99, San Jose, CA.

[107] O. Fatemi and S. Panchanathan, "Block Rotation: Implementation and Application", Proceeding of SPIE vol. 3166 pp 254-263, Parallel and Distributed Methods for Image Processing, July 97, San Diego, CA.

[108] O. Fatemi and S. Panchanathan, "Design Trends in Multimedia Hardware Architectures", Proceedings of SPIE Vol. 3311 pp 2-6, January 1998, San Jose, CA.

[109] O. Fatemi and S. Panchanathan, "Classification of Multimedia Processors", Media Processors 1999 January 1999, San Jose, California, Proceedings of SPIE vol. 3655 pp 135-146.

[110] O. Fatemi and S. Panchanathan, "Fractal Engine: An Affine video processor for Multimedia Applications", invited paper IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6 no. 7 pp 892-908, Nov. 1998