# ARCHITECTURES AND DESIGN TECHNIQUES FOR REAL-TIME IMAGE PROCESSING ICs

by

Peter Alexander Ruetz

Memorandum No. UCB/ERL M86/37

2 May 1986

# ARCHITECTURES AND DESIGN TECHNIQUES FOR REAL-TIME IMAGE PROCESSING ICs

by

Peter Alexander Ruetz

Memorandum No. UCB/ERL M86/37

2 May 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

**Architectures and Design Techniques for Real-Time Image-Processing ICs**

Ph.D. Peter Alexander Ruetz E.E.C.S.

## Abstract

A set of 8 chips, which perform real-time image processing tasks, was designed and fabricated with a 4μ NMOS technology. The chips include: a 3x3 linear convolver, a 3x3 sorting filter, a 7x7 logical convolver, a contour tracer, a feature extractor, a look-up-table ROM, and two post processors for the linear convolver. All chips were designed using architectures that are dedicated to the particular image processing task to be performed. The image processing circuits operate on 10 MHz video data (512 x 512 pixel images). The design time for the chips was kept to 1.5 man years by re-using hardware and, in addition, utilizing and developing some appropriate CAD tools. ROM generators and a data-path generator were developed to reduce the circuit design time. An image recognition system was built with these custom chips that can recognize two-dimensional objects that are characterized by their closed outer contours. The complete system is controlled by a SUN work station and operates at rates up to 15 $\frac{frames}{Sec}$. The recognition system achieved a 97% recognition rate for 8 objects over a wide range of orientation and size variations and a 100% recognition rate without size variations.

Robert W. Brodersen

Committee Chairman

# Table of Contents

Acknowledgements ..... iv
CHAPTER 1 INTRODUCTION ..... 1
1.1 Applications ..... 1
1.2 Computational Requirements ..... 2
1.3 Goals ..... 3
1.4 References ..... 5
CHAPTER 2 IMAGE PROCESSING REVIEW ..... 6
2.1 Introduction ..... 6
2.2 Segmentation ..... 6
2.3 Contrast Enhancement ..... 8
2.4 Filtering ..... 11
2.5 Noise Rejection ..... 13
2.6 Edge Extraction ..... 17
2.7 Edge Enhancement ..... 21
2.8 Binary Transformations ..... 21
2.9 References ..... 33
CHAPTER 3 IMAGE RECOGNITION TECHNIQUES ..... 35
3.1 Introduction ..... 35
3.2 Raw Data to be Utilized ..... 36
3.3 The System ..... 41
3.4 Recognition Features ..... 41
3.5 References ..... 47
CHAPTER 4 HARDWARE CHOICES ..... 48
4.1 Introduction ..... 48
4.2 Major Issues ..... 48
4.3 Comparisons of Image Processor Performance ..... 52
4.4 Dedicated Architectures for Space Efficient Processors ..... 63
4.5 References ..... 65
CHAPTER 5 DESIGN TECHNIQUES FOR FAST CIRCUIT DEVELOPMENT ..... 66
5.1 Introduction ..... 66
5.2 The Hardware Hierarchy ..... 66
5.3 CAD Issues ..... 67
5.4 Module Generation ..... 68
5.5 Data-Path Generation ..... 69
5.6 References ..... 91
CHAPTER 6 CIRCUITS ..... 92
6.1 Introduction ..... 92

6.2 Commonly Used Circuits ..... 92
6.3 Data-Path Cells ..... 94
6.4 Central Storage Cells ..... 102
6.5 References ..... 113
CHAPTER 7 THE IMAGE PROCESSING ICs ..... 114
7.1 Introduction ..... 114
7.2 Image Processor Interfaces ..... 126
7.3 A Single Chip 3x3 Convolver ..... 129
7.4 A 7x7 Logical Convolver ..... 142
7.5 A 3x3 Non-Linear Filter Based on Sorting ..... 148
7.6 An Image Contour Tracing Chip ..... 154
7.7 A Feature Extractor for the Image Contour Tracer ..... 163
7.8 The Line Delay ..... 173
CHAPTER 8 RECOGNITION RESULTS AND ANALYSIS ..... 181
8.1 Introduction ..... 181
8.2 The Features ..... 183
8.3 Recognizer Operation ..... 195
8.4 Recognition Results ..... 197
8.5 More Constrained Recognition ..... 198
8.6 Improving Recognition Accuracy ..... 203
8.7 Other Recognition Techniques ..... 204
8.8 Conclusions ..... 208
APPENDIX A THE RECOGNITION BOARD ..... 209
A.1 Descriptions of Chip Signals ..... 209
A.2 Board Layout ..... 214
APPENDIX B A COMPARISON OF STORAGE CELLS FOR DIGITAL SIGNAL PROCESSING ..... 226
B.1 Introduction ..... 226
B.2 General Tradeoffs ..... 226
B.3 Distributed Storage ..... 226
B.4 Centralized Storage ..... 227
B.5 Making Choices ..... 232
B.6 Design Considerations ..... 236
APPENDIX C NMOS CIRCUIT LAYOUT GUIDELINES ..... 237
C.1 Introduction ..... 237
C.2 Transistors ..... 237
C.3 Interconnect ..... 239
C.4 General Chip Layout ..... 241
C.5 Sensitivity ..... 241
APPENDIX D CAD INPUT DESCRIPTIONS ..... 243
APPENDIX E USING THE IMAGE PROCESSING SYSTEM ..... 264
E.1 The Graphical Interface ..... 264
E.2 Input and Output Formats ..... 267

APPENDIX F IMAGE PROCESSING PROGRAMS ..... 275

## Acknowledgements

My research advisor, Professor Robert Brodersen, provided me with excellent high level guidance. The advice he gave always seemed to be slightly ahead of its time. His technical and personal skills made it a pleasure to work with him.

I would like to acknowledge the value of the help and advice given by Robert Kavaler, also known as the roving consultant of Cory Hall. His knowledge of UNIX, the C programming language and his interest in just about everything made my research progress more rapidly.

The core graphics routines in "ittool" were developed by Brian Richards. This program made it possible to give good demonstrations in addition to providing a controller for the general development system.

This research was sponsored in part by DARPA and the General Electric Company. IBM partially supported me for three years as an IBM fellow.

The MOSIS group deserves a great deal of credit for providing a reliable and fast NMOS foundry service. They were also very helpful and cordial in trying to resolve any problems that arose.

Finally, my thanks go to my wife, Susan, who read many sections of this and tried to correct my numerous writing errors.

# CHAPTER I INTRODUCTION

Although work in image processing has been progressing for some time, it is beginning to accelerate with the advent of new image processors. These new processors attack the problems of I/O and computational bottlenecks that have long been associated with image processing. Hopefully, high performance, compact processors will make new image processing algorithms and systems feasible that were considered beyond the realm of possibility only a few years ago.

## 1.1 APPLICATIONS

There is a wide range of applications of image processing. Some broad classifications are image enhancement, restoration, coding, recognition and characterization. The goal in each of these applications is to convert an image either to a modified image (enhancement and restoration) or to some other representation that requires fewer bits of information (coding, recognition and characterization).

Image enhancement is the subjective improvement of an image for the human visual system. The goal is to allow a person to more easily see some particular feature or features of an image without regard for image fidelity. In fact, techniques are often used which purposefully and drastically alter an image. Medical X-rays are enhanced [1,2] to allow physicians to more easily locate tumors, broken bones or other conditions affecting the patient's health. Image enhancement is also used to help bring out details in or remove noise from images taken in outer space [3]. In these cases, where each image comes at considerable cost, it is desirable to extract as much information as possible.

Image restoration is concerned with restoring an image to its original state after the image has been degraded. In this case it is desired to maintain high image fidelity. One common example of image restoration is the correction of blurred images. The image blur could be caused by camera or subject motion [4], a turbulent atmosphere [5] or diffraction limited optics [6]. Unlike image enhancement algorithms

which tend to be ad hoc, image restoration algorithms are usually based upon more rigorous theory.

Coding of images is used to reduce the bandwidth required to transmit an image via satellite, telephone line or other medium. In addition, mapping satellites generate an enormous amount of data that must be stored. By reducing the transmission or storage requirements for the images, the cost of the image transmission and storage system is reduced. There are two broad classes of coders, those which produce and exact representation of the original image and those which do not. Usually, if some distortion in the final image is tolerable, the total number of the bits required to represent the image can be significantly reduced.

In an industrial environment, characterization of images is quite important. To ensure that manufactured items are of suitable quality, some kind of inspection must be performed. Normally, this is performed by a person who tends to be good at intuitive reasoning but has a more difficult time performing quantitative measurements or tedious tasks. In order to save on labor costs or to improve the reliability of performing tedious inspection tasks, the inspection may be performed with an image processing system. Automated inspection has been employed to verify the quality of thin film disk heads [7], to check that the placement of IC dies within a package is within specifications, and to inspect of various automotive parts [8].

Inspection deals with the characterization of a known object, while recognition deals with the determination of the identity of an unknown object. Recognition could be used to help a robot sort parts on a conveyer belt or to locate a part in order to perform some action upon it. With more advanced recognizers, robots could pick up articles around the house and put them away or look for an electrical outlet to recharge its batteries. In this field, the applications are virtually limitless for a robust 3-D natural scene recognizer.

## 1.2 COMPUTATIONAL REQUIREMENTS

There are many applications for image processing, but the computation requirements are quite severe. Images typically have a much higher data rate than speech or audio signals and hence special techniques must be employed to handle the I/O and computational bottlenecks.

A typical broadcast quality video image is sampled at a rate on the order of 10-20 MHz. This sampling rate will yield an image that is approximately 512 pixels on a side with a 30 Hz frame rate. It is important to note that the spatial and temporal resolutions both contribute to the high data rates. An image with large spatial resolution, 1K x 1K pixels, but very low temporal resolution, 1 frame/Sec, would only have to be sampled at a 1 MHz rate.

Audio band signals can be sampled at a much lower rate because audio signals are band limited to 20 KHz. For telephony applications, sample rates of 8 KHz are common while for digital audio a sample rate of 44 KHz is typically used. In contrast to video signals which are inherently three-dimensional functions (the dimensions are time, X, Y), audio signals are only one-dimensional functions (functions of time) and therefore there is no notion of a frame rate for audio.

The ratio of data rate to circuit clock rate is quite different for audio and video processing. For a standard MOS process, 10-20 MHz clock rates are feasible. Of course, the actual clock rate depends on the details of the processing technology, the type of signal processing that must be performed, the circuit design techniques and the circuit architecture. Assuming that the use of a 10 MHz circuit, the ratio, $R = \frac{F_{clk}}{F_{sample}}$, is 227-1250 for audio signals but only 1 for video signals. $R$ indicates how many times each piece of hardware can be used per sample to perform different operations in the algorithm. In the audio case, each piece of hardware can be used to perform hundreds of operations per sample, while in the video case, each piece of hardware can perform only one operation per sample. If $R$ is large, a single standard architecture can be micro-coded to perform a wide variety of tasks. However, as $R$ approaches 1, parallel architectures must be used, with one physical piece of hardware being used for each operation in the algorithm. To keep the processing elements and storage elements fully utilized, the processing elements have to be connected in a custom configuration.

## 1.3 GOALS

Several major goals guided this work. It was desired to produce high performance image processors that could perform many of the basic image processing functions that are needed in different applications. In addition, the decision was made to design all of the processors to operate in real-time. This

makes it possible to utilize all available data and hence achieve the highest possible throughput. In addition, by operating in real-time, the need for external frame buffers, which add to system complexity, cost and size is avoided. To demonstrate the ability of the chip set to perform some complete task, a goal was set to design and build an image recognition system. To make the problem feasible, the recognition goal was restricted to recognizing 2-dimensional objects that are characterized by their closed contours. Finally, there was a desire to keep the project development time to a minimum. The set of chips had to be designed, fabricated and assembled into a working recognition system in the time allotted for the Ph.D. degree.

Development of a real-time image processing system requires the merging of many fields. The designer or designers must have knowledge regarding the algorithms to be implemented, architectures and circuit design. To quickly develop custom circuits, some knowledge of CAD techniques is invaluable. Tradeoffs must be made between efforts in each of the fields. The designer must know how to choose new algorithms or modify existing algorithms to make the implementation easier. In addition, tradeoffs must be made between the circuit design effort and the CAD software effort. Essentially, if small compromises can be made in one area that result in a major savings of effort in another area, the designer will be successful.

Chapter 2 covers some basic image processing tasks, most of which were implemented on a chip. The functions implemented include 3x3 linear convolution, a 3x3 sorting filter operations, 7x7 logical convolution, contour tracing, feature extraction and point-wise non-linearities. These functions are common tasks that are mentioned often in the literature and these functions appeared to be sufficient to solve the recognition problem. In chapter 3, some basic recognition techniques are discussed. Some analysis is performed to determine the ideal performance that can be expected from these systems. A comparison of the performance, size and complexity of various types of hardware is made in chapter 4. In chapter 5, the techniques used to achieve a fast development time is covered. These include the use of hardware hierarchically and the appropriate use of CAD tools. The basic circuits used in many of the chips are covered in chapter 6. In chapter 7, the architectures and other details of the chips are discussed. Finally, in chapter 8, the performance of the recognizer is covered and some analysis is carried out to determine

the causes of recognition errors and ways in which the system could be improved. In the appendixes, details of the recognition board are covered (appendix A), some storage design techniques are discussed (appendix B) and the CAD input descriptions used in the chips are documented (appendix D).

## 1.4 REFERENCES

[1] Hall, E. L., et al., "A Survey of Preprocessing and Feature Extraction Techniques for Radiographic Images," *IEEE Trans. Computers*, vol. C-20, no. 9, pp. 1032-1044.

[2] Andrews, H. C., et al., "Image Processing by Digital Computer," *IEEE Spectrum*, vol. 9, no. 7, pp. 20-32.

[3] Nathan, R., "Picture Enhancement for the Moon, Mars and Man," *Pictorial Pattern Recognition*, G. C. Cheng, Ed., Thompson, Wash. D. C., 1968, pp. 239-266.

[4] Slepian, D., "Restoration of Photographs Blurred by Image Motion," *Bell Syst. Tech. J.*, vol. XLVI, no. 10, 1967, pp. 2353-2362.

[5] Horner, J. L., "Optical Restoration of Images Blurred by Atmospheric Turbulence Using Optimum Filter Theory," *Applied Opt.*, vol. 9, no. 1, Jan. 1970, pp. 167-171.

[6] Frieden, B. R., "Band-Unlimited Reconstruction of Optical Objects and Spectra," *J. Opt. Soc. Am.*, vol. 57, no. 8, Aug. 1967, pp. 1013-1019.

[7] Petkovic, D., et al., "An Experimental System for Disk Head Inspection," *IBM Research Report*, Dec. 1985.

[8] Baird, M. L., "Computer Vision Techniques for Locating and Determining Orientation of Circular Gear Banks," *GM Research Report CS*–244, Aug. 1978.

# CHAPTER II IMAGE PROCESSING REVIEW

## 2.1 INTRODUCTION

As discussed previously there are many applications for image processing. To implement an image processing system one must choose particular algorithms to perform various low level tasks such as noise rejection, edge enhancement, edge extraction, contrast enhancement and binary image transformations. Usually there are many different ways to perform each task. There are global and local techniques, linear and non-linear techniques and memory-less techniques.

To build a real-time image recognition system, image processing algorithms must be emphasized that are simple enough to implement quickly and efficiently in silicon. Further, it is important to choose algorithms that produce good results for machine decision making and not necessarily for subjective human viewing.

## 2.2 SEGMENTATION

There are two primary ways to segment an image into the foreground, or objects of interest, and the background. In the first technique, it is assumed that the background and foreground can be separated by using absolute gray-level information. In the second technique, it is assumed that there is a local discontinuity in gray values at the interface between the foreground and background.

### 2.2.1 Gray-Level Thresholding

In the first technique, it is assumed that objects in the foreground and background have sufficiently different gray levels so that the gray-level image can simply be thresholded. Pixels with gray values above the threshold (or below it) are considered to be in the foreground and pixels with gray values below the threshold (or above it) are assumed to be part of the background. If the image environment is

sufficiently well controlled, the threshold can be determined once and then left at this one particular value.

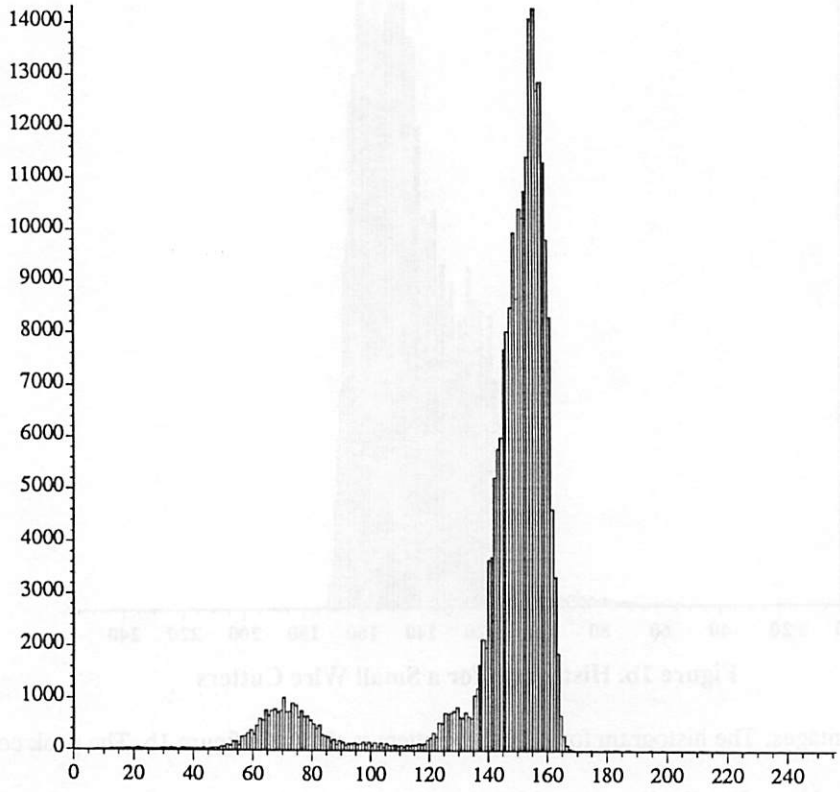



**Figure 1a. Histogram for a Large Wire Cutters**

For changing environments, the threshold is usually determined adaptively, using information from the image histogram [1,2]. If the background and foreground do have very distinct gray values, the histogram will have two distinct peaks in it; one corresponding to the background and one corresponding to the foreground.

Deficiencies in this technique begin to show up when the foreground and background gray levels overlap or when the foreground objects become small. An example of this is shown in figure 1a, the histogram for a pair of wire cutters. The two peaks can be clearly seen. The large peak at gray levels near 150 corresponds to the background, while the peak at gray levels near 70 corresponds to the foreground. The threshold was chosen to be half way between the two peaks and the resulting segmented image is shown in figure 6. This approach seems to have been fairly successful, except that a small piece of the cutters near the jaws was falsely classified as background. A greater problem arises in dealing with very

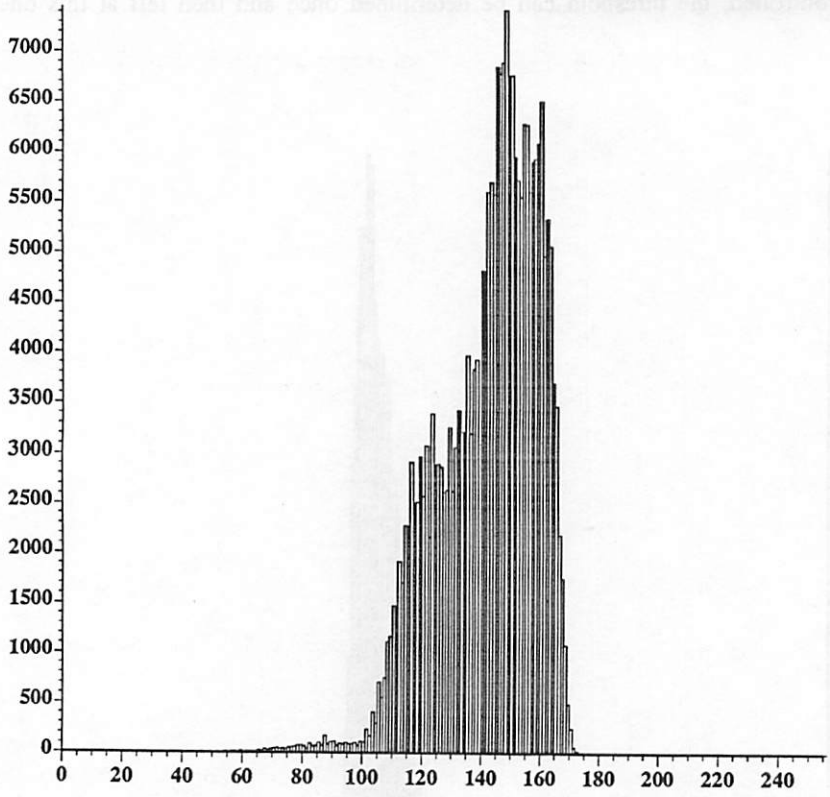



**Figure 1b. Histogram for a Small Wire Cutters**

small foreground images. The histogram for a smaller cutters is shown in figure 1b. The peak corresponding to the foreground has virtually disappeared, leaving little information to base the segmentation on.

### 2.2.2 Edge Extraction

Another common approach to image segmentation is involves finding "edges" in the image and assuming that the foreground and background are separated by these edges. These techniques will be discussed in greater detail later. The two cases examined above were also segmented with an edge extractor and the results are shown in figures 7a and 7b. The edge extractor generated a well defined boundary for both cases and hence has better size invariance, but the edge extractor also produced edges inside the object. For systems in which only the outer boundary information is utilized, this is not a problem. However, it may be undesirable in some systems.

## 2.3 CONTRAST ENHANCEMENT

Contrast enhancement [3] refers to the point-wise manipulation of grey levels to "improve" the

contrast of an image. This could be used to bring out details in medical X-Rays or to make an automated image recognition system less dependent on illumination variations. Because the operations are memory-less and the data is commonly only 8 bits, a simple way to provide an arbitrary grey level transformation is with a look-up table. This can be implemented in a RAM, ROM or PLA depending upon the nature of the problem.

If the image illumination is poor or variable, the grey levels of an image may span only a few bits and provide little contrast. Transforming the image by spreading these few bits into the entire range of available bits will subjectively improve the visibility of the image. The transfer characteristic can be chosen by a human operator or adaptively, using a technique know as histogram equalization [4,5] or modification. In this technique the histogram of an image, the relative probabilities of each grey level occurrence, is used to create a transfer characteristic to modify the original image. The output image has some desired histogram shape, usually flat, that utilizes the entire available range of grey levels. The idea can be demonstrated by assuming that the input (X) and output (Y) have continuous cumulative probability distributions, $F_X$ and $F_Y$ respectively.

If Y is computed from X by: $Y = F_X(X)$

Then:

$$F_Y(y) = Pr[Y<y]$$
$$= Pr[F_X(X)<F_X(x)]$$
$$= Pr[X<x]$$
$$= F_X(x)$$
$$= F_X(F_X^{-1}(y))$$
$$= y$$

so Y has a uniform probability density.

To summarize, the histogram equalized image is formed by modifying an image by a transfer characteristic that is equal to its cumulative probability distribution. If a non-uniform density is desired, another look-up table can be used to generate an arbitrary output image density.

All of this assumed that the probability densities are continuous. In reality, the densities are discrete which often results in fewer grey levels in the output image. This effect is most pronounced

when the histogram of the input image has strong peaks. Since it is not possible to map pixels with the same grey level to different output grey levels, the large peaks will exist in the output histogram but will be spaced out to make the average height of the histogram a constant. This technique can not add information which did not exist (i.e. more quantization values). For human viewing, the reduction in the number of grey levels in the image may not be a problem (if the number of grey levels is reduced too much contouring of the image becomes obvious) and the resulting image will often be seen as improved. However, problems occur when edges are extracted from the equalized image because pixels that originally were very close in grey value near the peak of the histogram become much farther apart in the output image. Therefore differences in grey level that were caused by low level noise will be amplified and result in false edges.

The problem can be illustrated by assuming that the input and output images have continuous gray levels. The common method of edge detection is to find spatial gradients. If the spatial gradient is large, the point is called an edge point. So, if $Y = F_X(X)$ and the edge extraction filter is a first derivative in the "s" spatial direction, then:

$$y_{edge} = \frac{dy}{ds} = \frac{dF_X(x)}{ds} = \frac{dF_X(x)}{dx}\frac{dx}{ds} = f_X(x)x_{edge}$$

Therefore, differentiating the histogram equalized image is equivalent to weighting the derivative of the original image by the pdf. It can now be clearly seen that even if $x_{edge}$ is small and possibly non zero only because of noise, $y_{edge}$ can be large and a false edge will be found.

The problems with false edges can be seen in figures 2 and 8. Figure 8a shows the original edge map image and the histogram of the grey level image is shown in figure 2a. Notice that there is a strong peak in the histogram that corresponds to the nearly uniform background. The histogram of the equalized image is shown in figure 2b and the corresponding edge map in 8b. The pixel values from the peak of the original histogram (128-160 grey values) were spread over almost the entire range (40-255). The result of this can be seen in the edge map. The background is full of false edges due to to gain introduced by the equalization.

Histogram equalization can be performed with a few simple operations. First $f_x(x)$ must be computed by counting the number of pixels that have grey level $x$. $F_x(x)$ is then computed by performing the

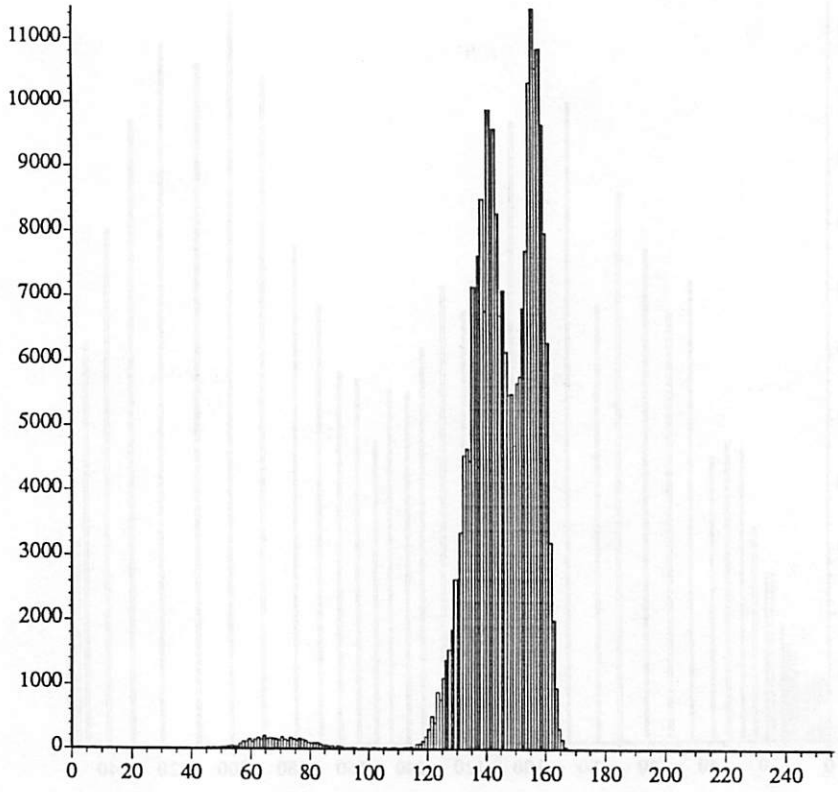



**Figure 2a. Histogram of Original Image**

discrete integral: $F_x(x) = \sum_{i=0}^{x} f_x(i)$

Another technique is to simply normalize the image by a linear to logarithmic transformation. This is the basis of homomorphic processing [6] and has advantages by reducing the sensitivity of future processing to the image illumination. This technique also introduces gain for small grey values and, therefore, also has some of the same problems as histogram modification. In fact, the logarithmic conversion is similar to histogram equalization when the the image has a histogram that is peaked for small grey values (they are identical if the image has a probability density $f_X(x) = \frac{1}{x}$).

## 2.4 FILTERING

The noise rejection, edge extraction and edge enhancement algorithms are all based upon linear or non-linear filters. The linear filters can be FIR (finite impulse response) or IIR (infinite impulse response, or recursive). The difference functions for the FIR and IIR filters are:

$$g_{x,y} = \sum_{n}\sum_{m} h_{n,m} f_{x-n,y-m}$$

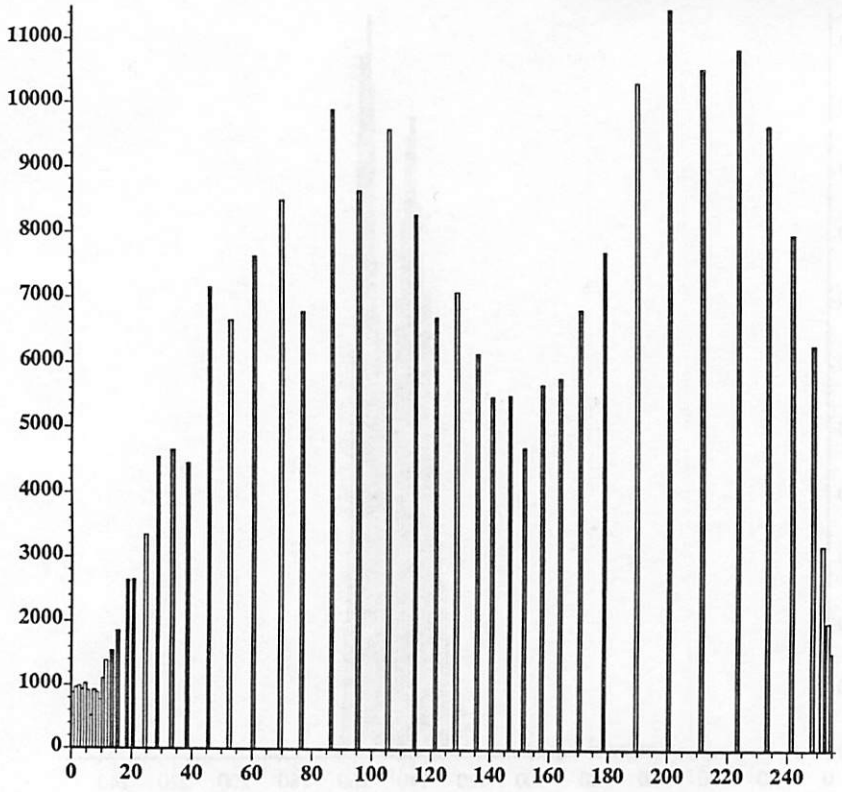



**Figure 2b. Histogram of Histogram Equalized Image**

$$g_{x,y} = \sum_{n}\sum_{m} h_{n,m} f_{x-n,y-m} + \sum_{n>0} \sum_{m>0} a_{n,m} g_{x-n,y-m}$$

The FIR filters are commonly used for several reasons. First, FIR filters can be easily pipelined to reduce the required performance of the circuits. Although IIR filters can be pipelined, pipelining requires a significant increase in complexity. Also, since good image models typically do not exist, ad hoc filtering techniques tend to be used that can be implemented with short FIR filters. FIR filters with short impulse responses (3x3 or shorter) have been commonly discussed[7]. Template matching [20] can be performed with an FIR filter where the filter impulse response is the template. These are popular because the short impulse responses can be implemented with less hardware and hence at a lower cost while larger impulse response filters can be made from a cascade of 3x3 impulse response filters. As higher density circuits become possible, the impulse response length will likely increase.

In addition, it is very common for the impulse responses coefficients to be only powers of two [8,12,14,15]. Again, this reduces the cost of the circuit by making it possible to use a small multiplier. For very short impulse response lengths, it is usually not necessary to have higher precision coefficients

(see low-pass filter example later) because careful tailoring of the frequency response is not possible.

The frequency response of the FIR filter can be found by taking the Fourier transform of $h_{n,m}$:

$$H(\omega_x,\omega_y) = \sum_n \sum_m h_{n,m} e^{-j(n\omega_x + m\omega_y)}$$

$$H_X(\omega) = H(\omega_X,0) = \sum_n h_n e^{-jn\omega_x} \qquad where \quad h_n = \sum_m h_{n,m}$$

$$H_X(\omega) = h_0 + 2\sum_{n>0} h_n cos(n\omega) \qquad if h_n = h_{-n}$$

$$H_X(\omega) = h_0 + 2j\sum_{n>0} h_n sin(n\omega) \qquad if h_n = -h_{-n}$$

## 2.5 NOISE REJECTION

Noise rejection is a common image processing task. Often, the point of view is taken that in a controlled environment, a sufficiently high SNR can be obtained so that noise rejection processors will not be needed. This is true for the human visual system which is insensitive to most kinds of noise. However, in systems which employ local operators to extract edges (i.e. high-pass filters), even seemingly imperceptible noise can produce an output that is comparable with the "true edges".

There are two broad categories for low-pass filters for noise rejection: linear and non-linear.

### 2.5.1 Linear Low-Pass Filters

Some examples of 3x3 FIR low-pass filter impulse responses [8] (neglecting the gain factor required for unity DC gain) are shown in table 1.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 2 | 4 | 2 | 1 | 2 | 1 |
| 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| (a) | | | (b) | | | (c) | | |

**Table 1. Impulse responses for linear FIR low-pass filters**

Note: filters (a) and (b) are separable and can be represented as by impulse responses in the vertical and horizontal dimensions as shown in table 2.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 1 | | | | 1 | |
| 1 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | |
| | | | 1 | | | | 1 | |
| (a) | | | (a) | (b) | | | (b) | |

**Table 2. Corresponding Vertical and Horizontal Impulse responses for filters 1a and 1b**

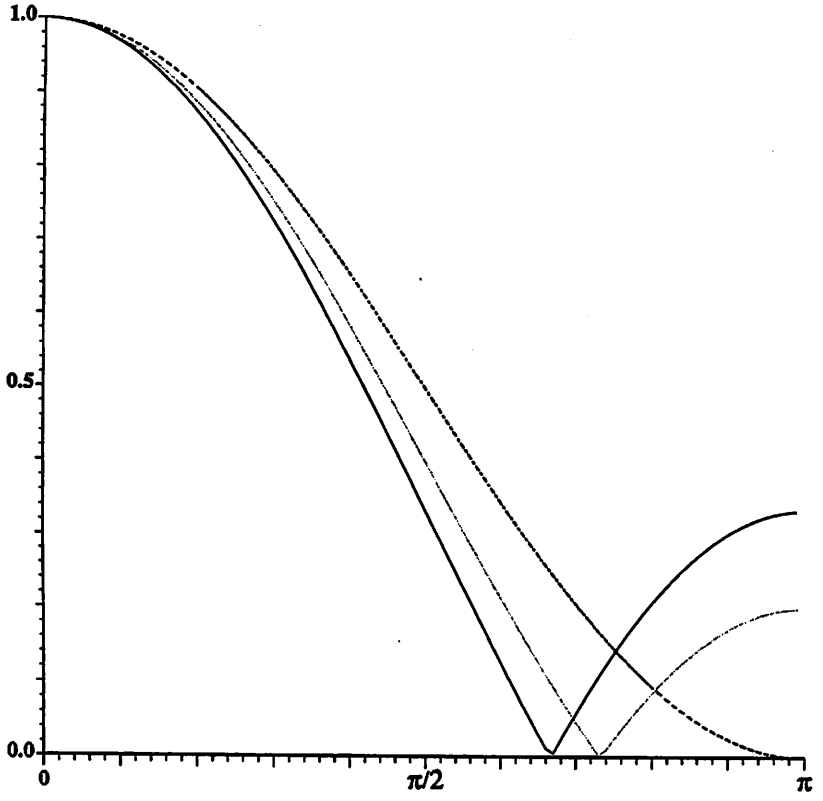



**Figure 3. $|H_X(\omega)| = |H_Y(\omega)|$ for filter a (solid), filter c (dotted) and filter b (dashed) in table 1.**

Visually, these filters appear to do very little. A slight blurring of the image can be seen which does not appear to be significant. However, if the low-pass filtering is performed before high-pass filtering and thresholding for edge extraction, an improvement in the resulting edge map can be seen (fewer edges caused by noise).

All of the filters perform similarly. The reason for this is that the shape of the frequency responses for each filter is very similar (see figure 3 for a plot of the one dimensional frequency responses). The filters in table 1 have the following impulse responses (for $\omega_Y = 0$):

$$H_X(\omega) = \frac{1+2\cos(\omega)}{3}$$

$$H_X(\omega) = \frac{1+1\cos(\omega)}{2}$$

$$H_X(\omega) = \frac{2+3\cos(\omega)}{5}$$

With such a short impulse response significant attenuation occurs only at fairly high spatial frequencies ($>\frac{\omega_s}{4}$). Even though the coefficients of the filters are coarsely quantized to powers of two, the frequency responses are virtually identical. This indicates that more precision in the coefficients is not a necessity to a shaper roll off. A longer filter impulse response would be needed to produce a narrower filter.

### 2.5.2 Non-Linear Low-Pass Filters

If linear filters are used for noise rejection preceding high-pass filtering for edge extraction, the edges will be blurred along with the noise. If the filters are not carefully tailored to the power spectrums of the noise and the image (as is not the case in 3x3 operators) the low-pass filter will simply make it "harder" to find the edges. To combat this problem, non-linear filters are often used for noise removal. The most common non-linear filter for noise removal is the median filter, which takes the median of a neighborhood as its output. Another technique rejects "out of bounds" pixels. Both of these filters tend to perform well on noise that has a probability density with long tails (pulse type noise) while the linear filter performs well on Gaussian noise.

The "out of bounds" filter [9] is based upon a linear filter. To determine if a pixel is "out of bounds", it is compared to the average of the 8 pixels around it. If the difference is larger than a predetermined threshold, the pixel is replaced by the local average. With this technique, pixels are only changed when detected to be "bad". If the threshold is chosen properly, obvious errors (pulses) will be removed but edges will be well preserved. The problem is that the threshold must be determined and the optimum value depends upon the image and noise characteristics. If the threshold is too high, noise will not be rejected properly. A threshold that is too low will cause the filter to become linear with the disadvantages described earlier. With a human operating in the loop, it can be chosen subjectively. Otherwise, it must be chosen a priori or an algorithm must be developed to adaptively choose the threshold.

Probably the most common type of non-linear filter for noise rejection is the median filter [10]. The median filter has some advantages over the filters discussed previously. First, edges or other monotonic changes (over a distance greater than the filter length) are preserved while pulse type noise is removed. Unlike the "out of bounds" filter, no threshold is needed for proper operation.

The major disadvantage of median filters is the computational requirements. To sort the elements over an $n \times n$ region requires at least $n^2 \log_2(n^2)$ comparisons (bubble sorters use $\frac{n^4}{2}$ comparators to achieve a more regular circuit layout). For filters operating over a large region, the median filter becomes much more costly than linear filters which require only $n^2$ multiplies and additions. To reduce the computation costs of the median filter it has been proposed to use a separable median [1, chap 7] instead of a true two-dimensional median. The separable median is found by utilizing a cascade of two median processors, one operating in the vertical dimension and one in the horizontal dimension. The total number of comparisons required is reduced significantly from 29 to 6 (for n=3). The separable median has somewhat poorer noise rejection properties but still rejects pulse type noise well.

A second problem with the median filter is that, unlike the linear filters, it tends to give the image an unnatural appearance. However, for image recognition systems, the subjective appearance of the image is of no consequence. The only thing that is important is the overall performance of the system.

Filters which replace a pixel by the minimum or maximum of some region around the pixel have been proposed for use as noise rejection filters [11].

The effectiveness of some of these noise rejection filters was tested by putting the noise rejecter before an edge extractor and examining the edge map for different types of image noise. Figure 9a shows the edge map of a scene with no noise and no noise rejection filters. The edge map of the image with additive Gaussian noise is shown in figure 9b (again no noise rejection was used). When the rectangular impulse response low-pass filter is used, the edge map of figure 9c results. Many of the false edges caused by the noise pulses cleared have disappeared without major degradation of the edges. When a 3x3 median filter is used more false edges were left (figure 9d). For pulse noise, the median filter performs far better than the linear filter. The edge map of an image with added pulse noise (and no noise

rejection) is shown in figure 9e. With median filtering (figure 9f), the noise is almost completely removed. The edge map of the linear filtered image (figure 9g) is actually worse than that with no noise rejection as the pulses were smeared over a wider region.

## 2.6 EDGE EXTRACTION

Because many image processing systems use the edges of the image for segmentation, characterization or recognition, much work has been done to develop filters to perform edge extraction. Here the emphasis is on operators that produce outputs that can be used by a machine to make decisions, not on operators that produce good subjective results for a human observer. Again, there are both linear and non-linear techniques and variation of the size of the region that is utilized.

### 2.6.1 Linear Edge Extraction Filters

There are several very common 3x3 FIR filters that have been examined and characterized for edge extraction. The masks of some of these are shown in table 3.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -1 | 0 | -1 | -1 | -1 | 1 | 2 | 1 | 1 | 0 | -1 |
| -1 | 4 | -1 | -1 | 8 | -1 | 0 | 0 | 0 | 2 | 0 | -2 |
| 0 | -1 | 0 | -1 | -1 | -1 | -1 | -2 | -1 | 1 | 0 | -1 |
| (a) | | | (b) | | | (c) | | | (d) | | |
| 1 | 1 | -1 | -1 | 1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 |
| 1 | -2 | -1 | -1 | -2 | 1 | 1 | -2 | 1 | 1 | -2 | 1 |
| 1 | 1 | -1 | -1 | 1 | 1 | -1 | -1 | -1 | 1 | 1 | 1 |
| (e) | | | (f) | | | (g) | | | (h) | | |

**Table 3. 3x3 edge extraction filters**

All of these filters are high-pass filters (the DC gain is the sum of all coefficients which is zero for all the cases shown); this is a basic property of edge extraction filters. To determine if a point is an edge point, the magnitude of the output of one of these filters is usually thresholded. Large filter outputs correspond to edges while small filter outputs do not.

Unlike the linear low-pass filters, the filters in table 3 perform quite differently and have quite different frequency responses. Filters 3a and 3b are omni-directional high-pass (Laplacian [12]) filters with the following frequency responses:

$$H_X(\omega) = H_Y(\omega) = 1-\cos(\omega)$$

Pratt [13] proposed a measure of performance for these filters that penalizes a filter for responding to noise. The Laplacian filters do not perform well because they have peaked responses along each frequency dimension (figure 4) at normalized frequency $\pi$. This corresponds to very high frequency signals so these filters enhance the noise greatly. Just by looking at the impulse responses one can see that these filters produce their maximum output when pulse noise is encountered (when the center pixel is bright while the background is dark). In fact, these are the filters that are used to indicate that a pixel is "out of bounds" in the non-linear low-pass filter.

Filters 3c (Sobel Y) and 3d (Sobel X) [14] have frequency responses (below and figure 4) that are quite different from the Laplacian filters. The major difference is that the Sobel filters are separable and are band-pass filters in one dimension and low-pass filters in the other.

Sobel Y: $H_X(\omega) = 1+\cos(\omega)$

Sobel Y: $H_Y(\omega) = \sin(\omega)$

Sobel X: $H_X(\omega) = \sin(\omega)$

Sobel X: $H_Y(\omega) = 1+\cos(\omega)$

These Sobel filters have the separable impulse responses shown below:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | 1 | | | | 1 |
| 1 | 2 | 1 | 0 | 1 | 0 | -1 | 2 |
| | | | -1 | | | | 1 |
| (a) | | | (b) | (c) | | | (d) |

**Table 4. Separable Impulse responses for the Sobel X (c,d) and Y (a,b) filters**

The Sobel filters also perform quite differently (have better edge extraction characteristics) than the Laplacian filters. The primary reason is that the Sobel filters are low pass in one dimension. Further, the

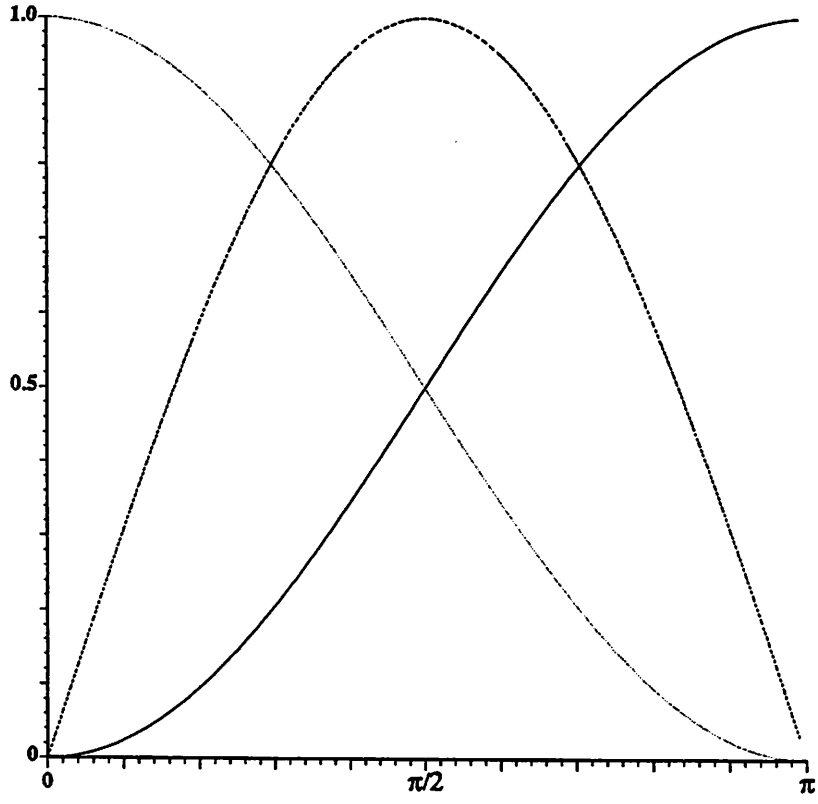



**Figure 4. Frequency Responses for the Laplacian and Sobel Filters. The dotted and dashed curves are the responses of the Sobel filters in the $\omega_X$ and $\omega_Y$ directions. The Laplacian filter (solid) has the same response in both dimensions.**

band-pass filter of the Sobel filters has a steeper rise at low frequencies and falls off for high frequencies. The Sobel X and Y filters are fairly insensitive to noise because they are low-pass filters in one dimension. This means that the filters will not respond to the high frequency signals (such as noise) at half the sample rate. Because the low-pass filter operates perpendicularly to the high-pass filter, the output of the filter will be large only if the edge exists over several pixels, as opposed to the omni- directional filters which have maximum output for an isolated point.

When using the Sobel X and Y filters, the edge map is obtained by taking the square root of the sum of the squares of the Sobel filters which approximates the magnitude of the gradient at that point. Commonly the sum of absolute values is performed instead of the square root of sum of squares to simplify the computations.

The remaining four filters (3e - 3h) are from a set of eight compass gradient filters [15]. Each filter produces its maximum output when an edge occurs in the same orientation as the filter impulse response

(there is one filter for every multiple of 45d). In this way, the direction in addition to the magnitude of an edge can be determined. This is useful in schemes that use relaxation as well as the magnitude and direction of edges to determine whether marginal edges should be rejected or not [21]. Essentially, in each pass of the algorithm, for each point in the image, the confidence of that point being an edge point is modified. The confidence in the previous iteration and the direction of nearby edge points and the confidence that neighboring points are edge points are all used to modify the confidence that the current point is an edge point. This procedure continues until a steady-state solution is reached. For example, if the current point has a low confidence of being on an edge (the magnitude of the high-pass filter output is small), but adjacent points have similar edge directions and high confidence values, the confidence that the current point is an edge will be increased.

To achieve better immunity to noise it may be necessary to use longer filter lengths and more complex filtering schemes. One suggestion [16] is not to use a single filter but many filters of different lengths. An appropriate algorithm is used to correlate the filter outputs to better reject noise and find edges that shorter impulse response filters can not. Others have suggested the use of "best fit" algorithms [17] to achieve noise immunity.

### 2.6.2 Non-Linear Edge Extraction Filters

A non-linear edge extraction filter that is based upon sorting, like the median filter, is the maximum difference operator. It finds the maximum difference of pixel values in a region and has high-pass characteristics. Although this filter will produce a maximum output for pulse noise, it responds better to edges (produces a higher output at human determined edges) than the Laplacian filter so that the noise performance is better than the Laplacian filter. In fact, subjectively, it appears to perform very similarly to the Sobel filter.

There are several advantages of the maximum-difference filter in implementation. The arithmetic units need only be as wide as the data words to retain full precision. Further, comparators are the basic computational elements which are more compact than multiplying adders. Unlike the median filter, the separable maximum difference is identical to the true two-dimensional maximum difference so that no concessions are made by reducing the number of comparators used to implement a separable filter.

Further, the maximum and minimum can be computed with only $4(n-1)$ comparators compared to $n^2$ multiplying accumulators need for a *nxn* convolution.

A comparison of the operation of edge extractors based upon the Laplacian, Sobel, maximum difference and compass gradient filters is shown in figure 10. Figure 10a is the edge map after Laplacian edge extraction. Note the false edges due to noise and the weak real edges. The Sobel edge map (figure 10 b) has many fewer false edges and much stronger real edges. The edge map generated by the maximum difference operator (figure 10c) looks similar to the Sobel edge map except there are slightly more false edges. Finally, the compass gradient edge map (computed by thresholding the sum of the magnitudes of the north and east operators) is shown in figure 10d. It appears to a have fewer false edges than the Sobel edge map and also fewer real edge points.

## 2.7 EDGE ENHANCEMENT

Edge enhancement is similar to edge extraction in that processing is done to make the edges more obvious. However, edge enhancement is only concerned with improving the subjective quality of a image for a human observer. Filters similar to those used in edge extraction can be used for edge enhancement. By adding the high-pass output of the Laplacian filters back to the original, the edges are made more prominent. By varying the amount of high-pass signal that is added to the original, the amount of edge enhancement can be varied.

## 2.8 BINARY TRANSFORMATIONS

Just as filters are utilized to perform transformations on grey level images, they are also used to transform binary images. The most common operations that are performed are dilation and erosion.

Dilation is the expansion of the foreground and is comparable to low-pass filtering in that fine image detail is lost. There are many uses of dilation [18,19] including: edge joining, bandwidth reduction, closing and opening. In our image recognition system, dilation is used to connect broken edges in the edge maps because the contour tracer needs continuous contours. Further, dilation is employed to reduce the bandwidth of the binary image to allow down-sampling without aliasing.

The dilation operator can be thought of as an FIR filter that operates on 1-bit input images with 1-bit impulse responses and produces 1-bit output images. If both the input data and coefficients are single bit, the multiplication of the convolution equation becomes an AND operation. If the sum is saturated to a single bit, the sum can be replaced by an OR operation.

Therefore the difference equation for dilation is:

$$g_{x,y} = \underset{m,n}{OR}(h_{n,m}\ AND\ f_{x-n,y-m})$$

Because all of the coefficients are of the same sign (0 or 1), the filter is basically low pass. Just as in multi-bit filters, the longer the mask or "impulse response", the less detail gets through.

Erosion is the expansion of the background and can be found by inverting the input image, dilating and inverting the output image.

Closing is performed by first dilating then eroding, while opening is performed by eroding followed by dilating. Closing fills in all portions of the image that are not large enough to hold the mask. Opening only leaves regions that are large enough to hold the mask.

The operation of dilation is illustrated in figure 11. Figure 11a shows the edge map without dilation. Note that many edges have small breaks in them. In figure 11b the edge map has been dilated by 1 pixel (mask shown in figure 5). The small gaps have been closed in. After 3 pixel dilation (figure 11c), the edges are all well defined but fine details have been lost.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| | | | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| | | | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| (a) | | | (b) | | | | | | |

**Figure 5. Masks for 1 pixel dilation (a) and 3 pixel dilation (b)**

processing examples

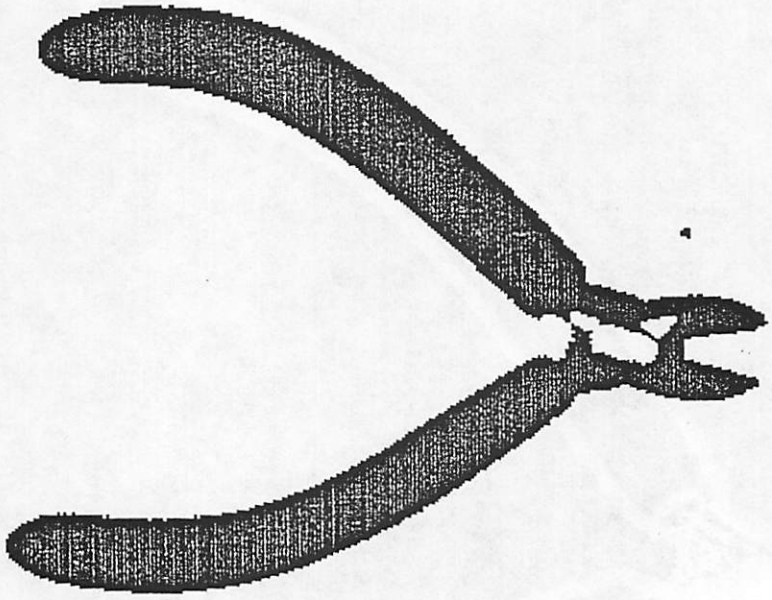


**Figure 6. Thresholded Gray-Level Image**

processing examples

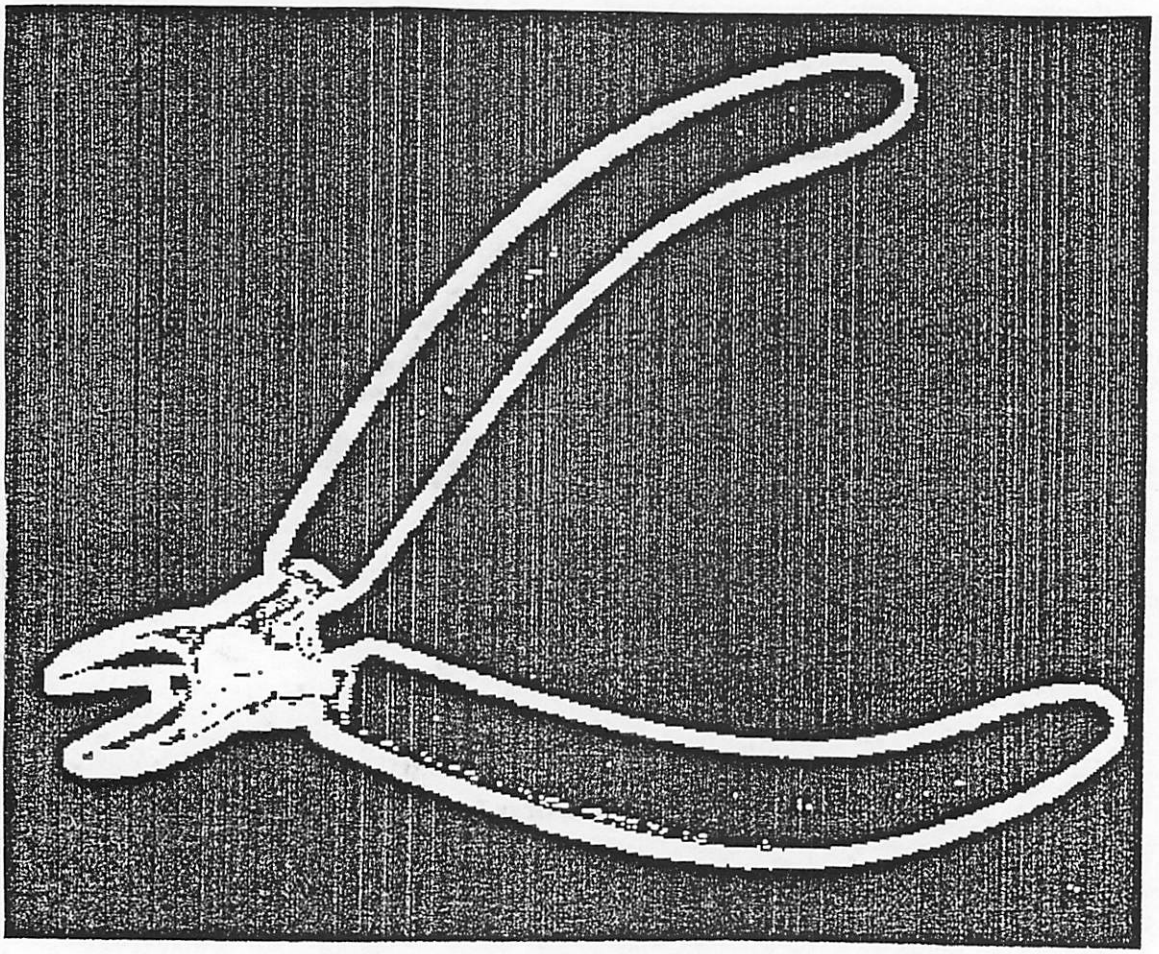


Figure 7a. Edges Extracted, Large Image

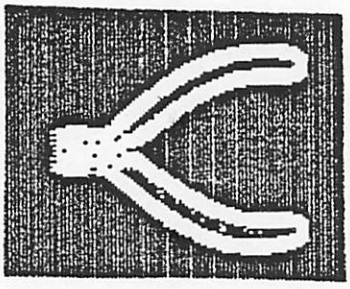


Figure 7b. Edges Extracted, Small Image

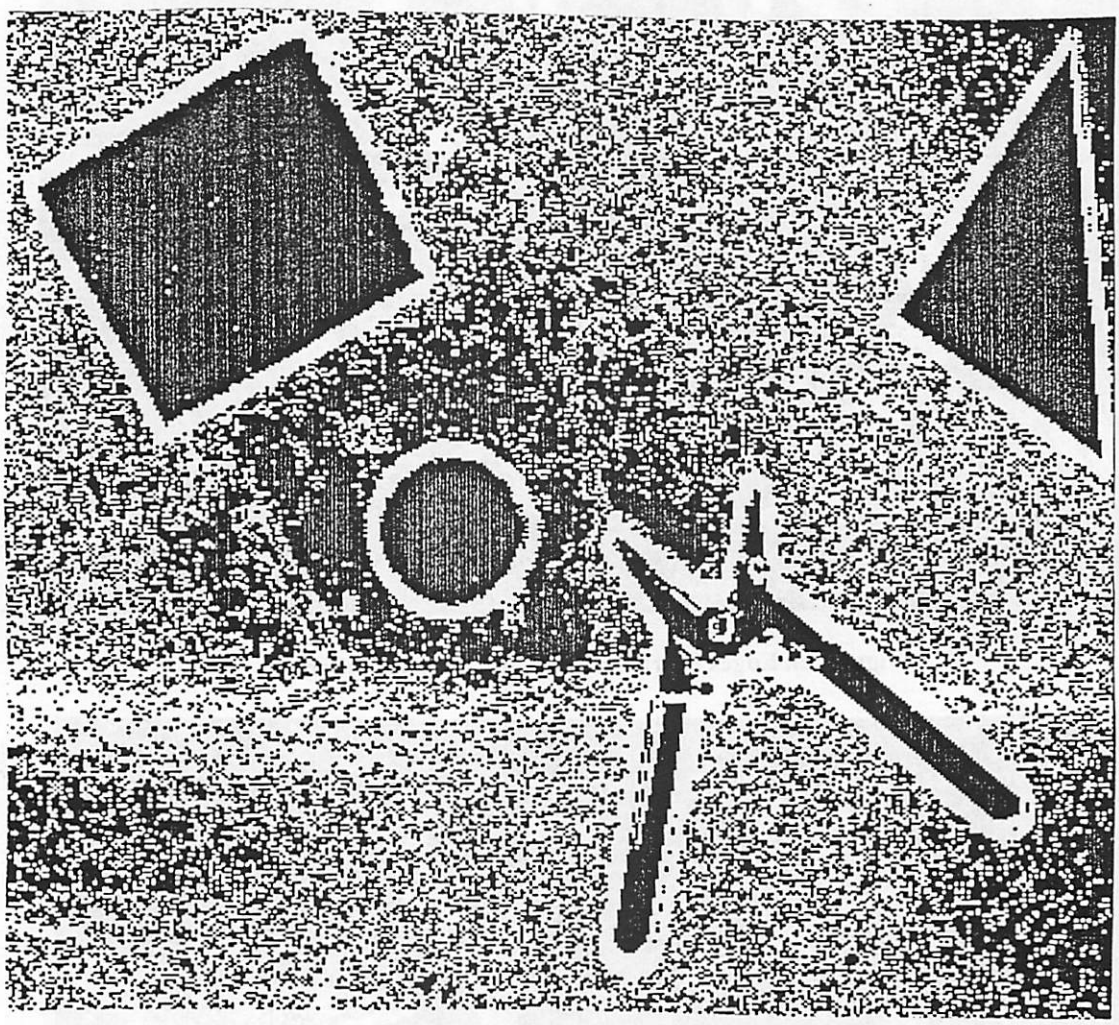


Figure 8b. Edges Extracted, Histogram Equalized Image

processing examples

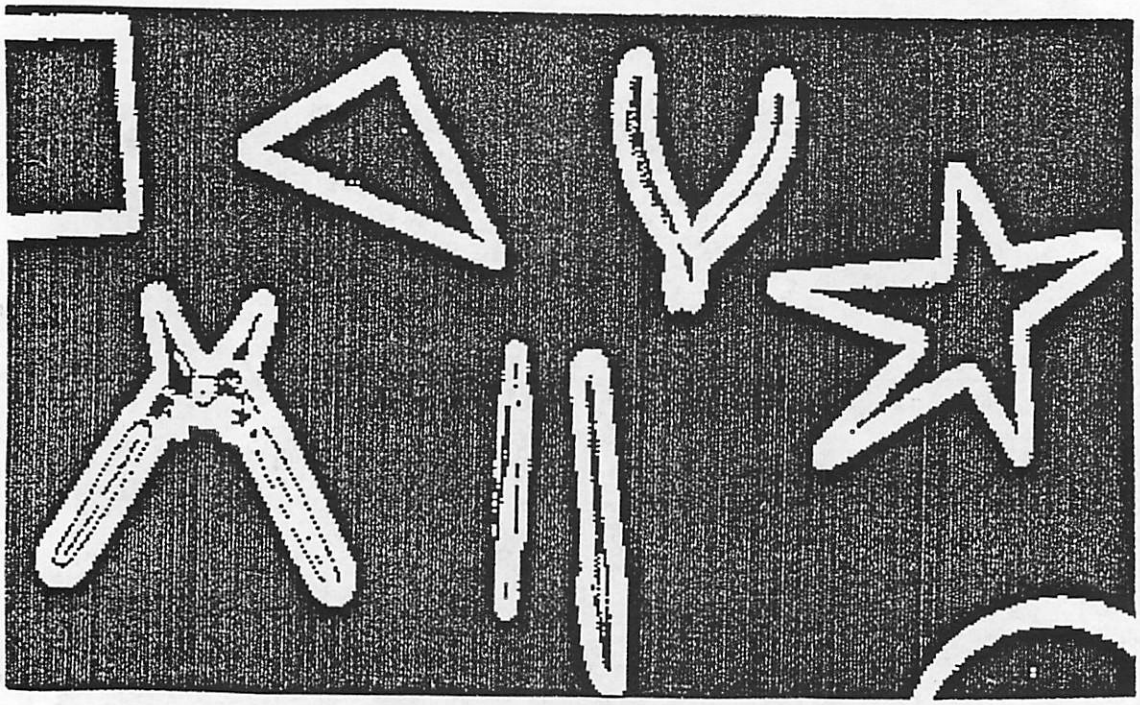


Figure 9a. Edges Extracted, no Noise or Noise Filter

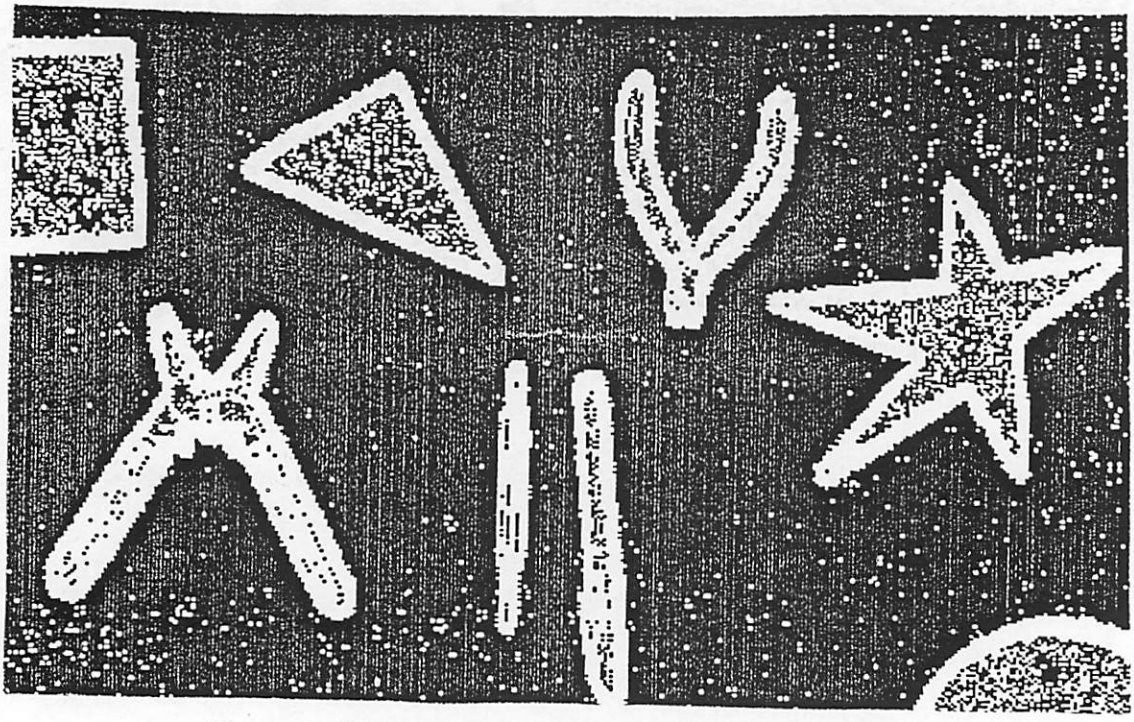


Figure 9b. Edges Extracted, Gaussian Noise and no Noise Filter

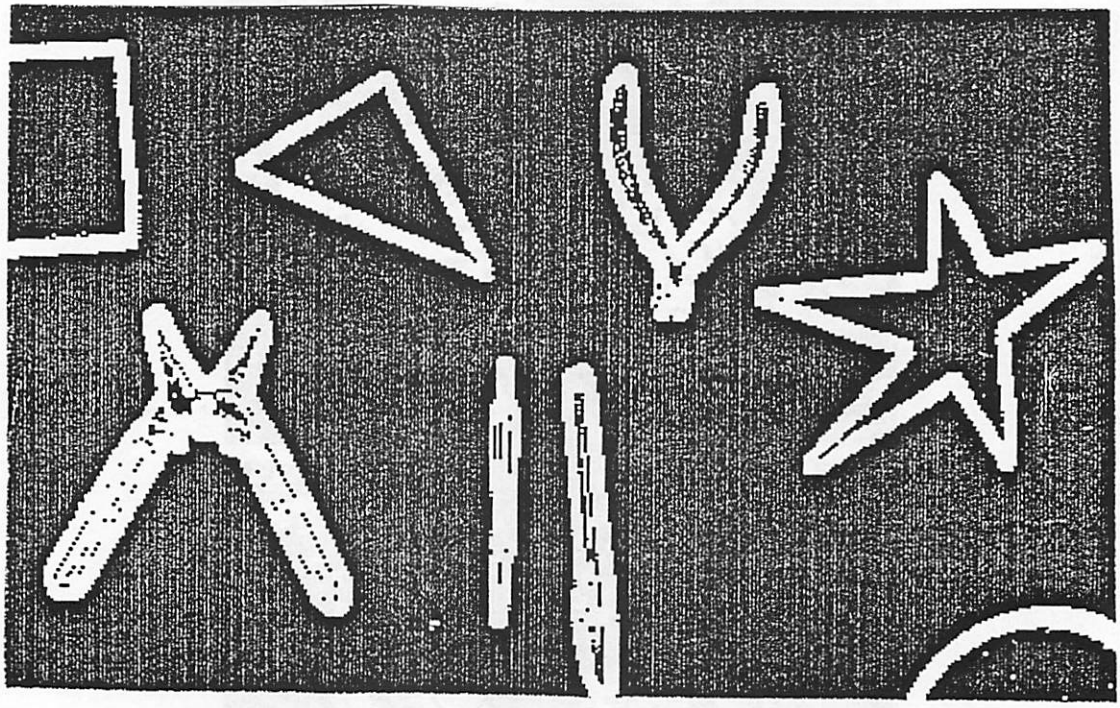


Figure 9c. Edges Extracted, Gaussian Noise and Linear Noise Filter

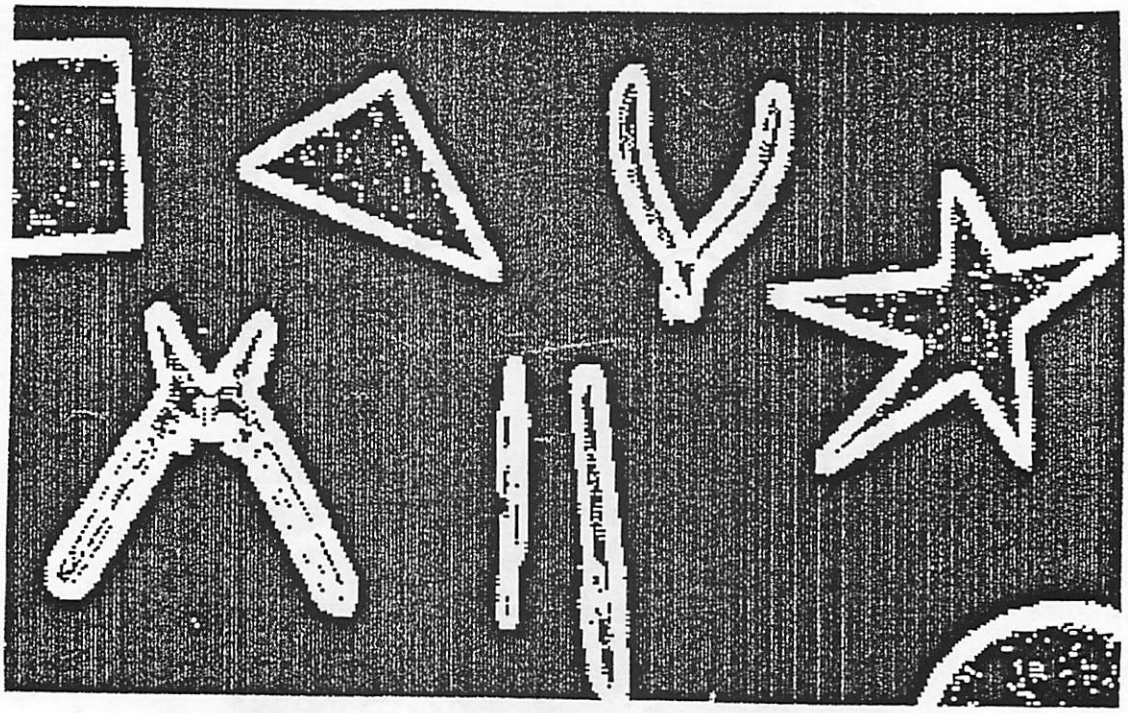


Figure 9d. Edges Extracted, Gaussian Noise and Median Noise Filter

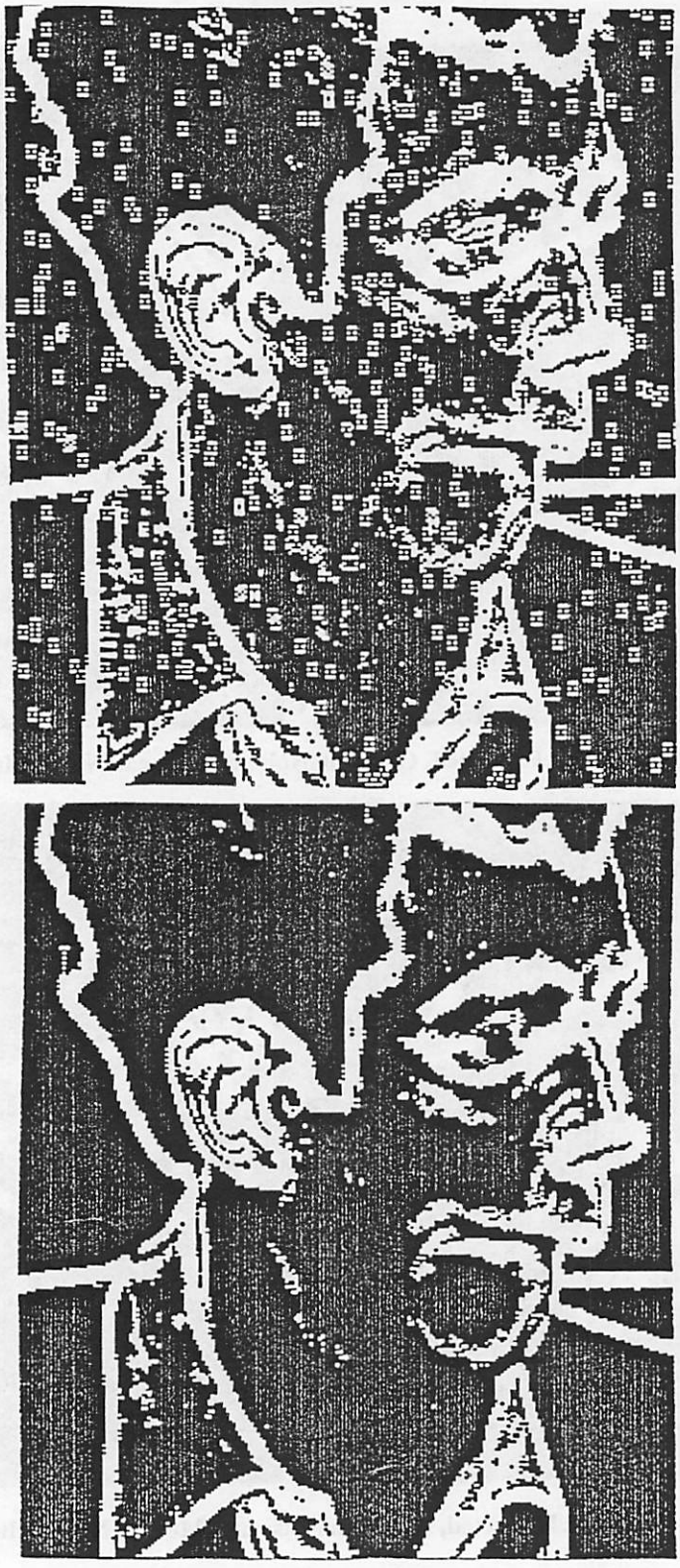


Figure 9e,f. Edges Extracted with Pulse Noise, No Filter (top), Median Noise Filter (bottom)

processing examples

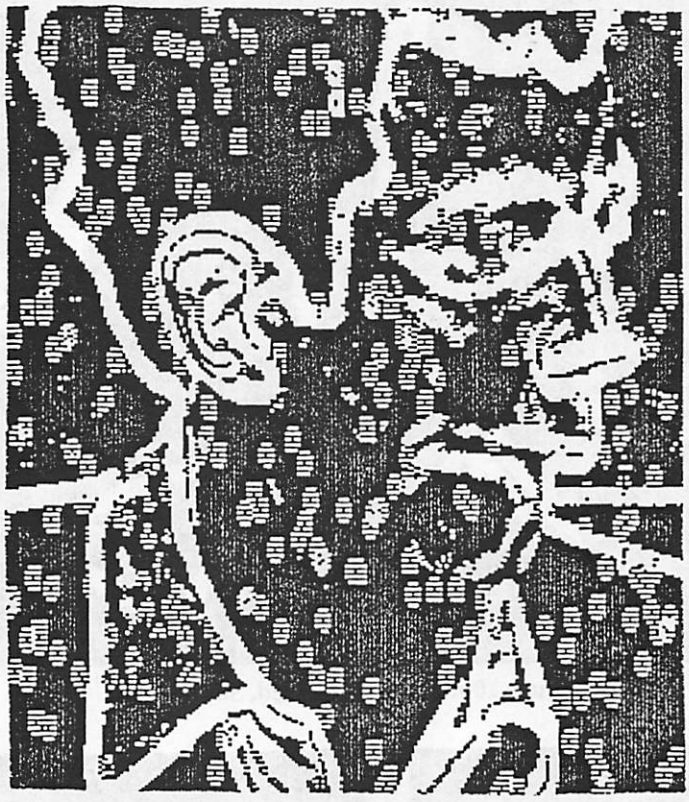


Figure 9g. Edges Extracted with Pulse Noise and Linear Noise Filter

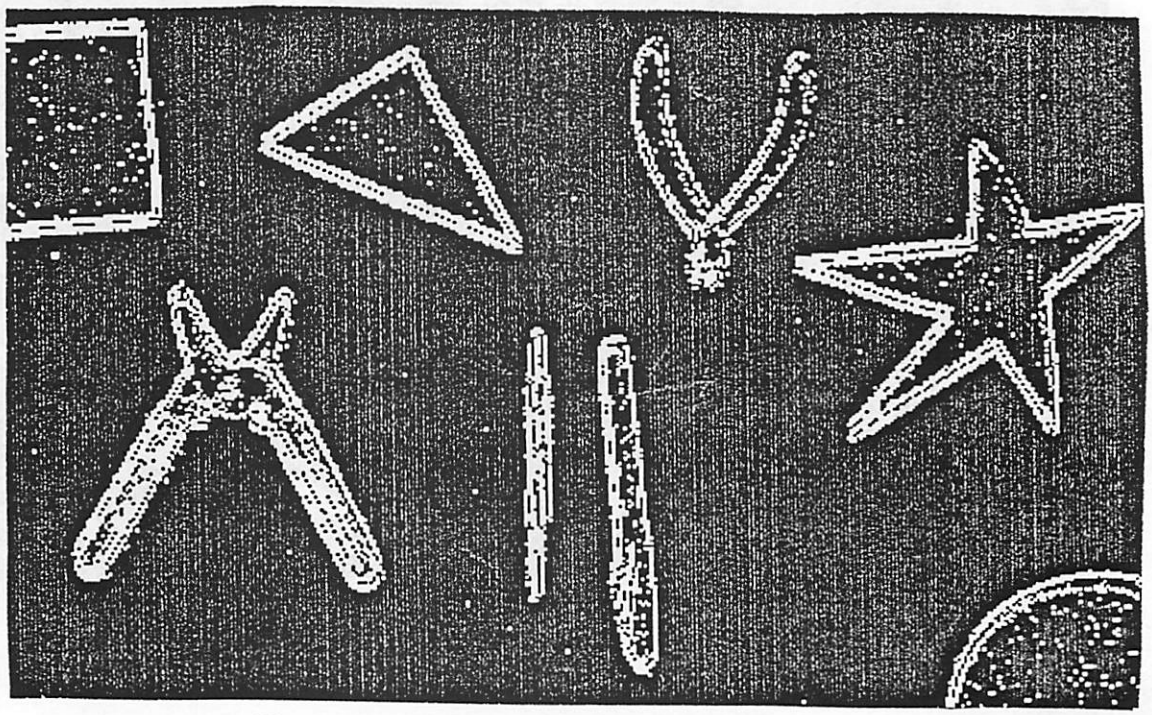


Figure 10a. Edges Extracted, Laplacian Filter

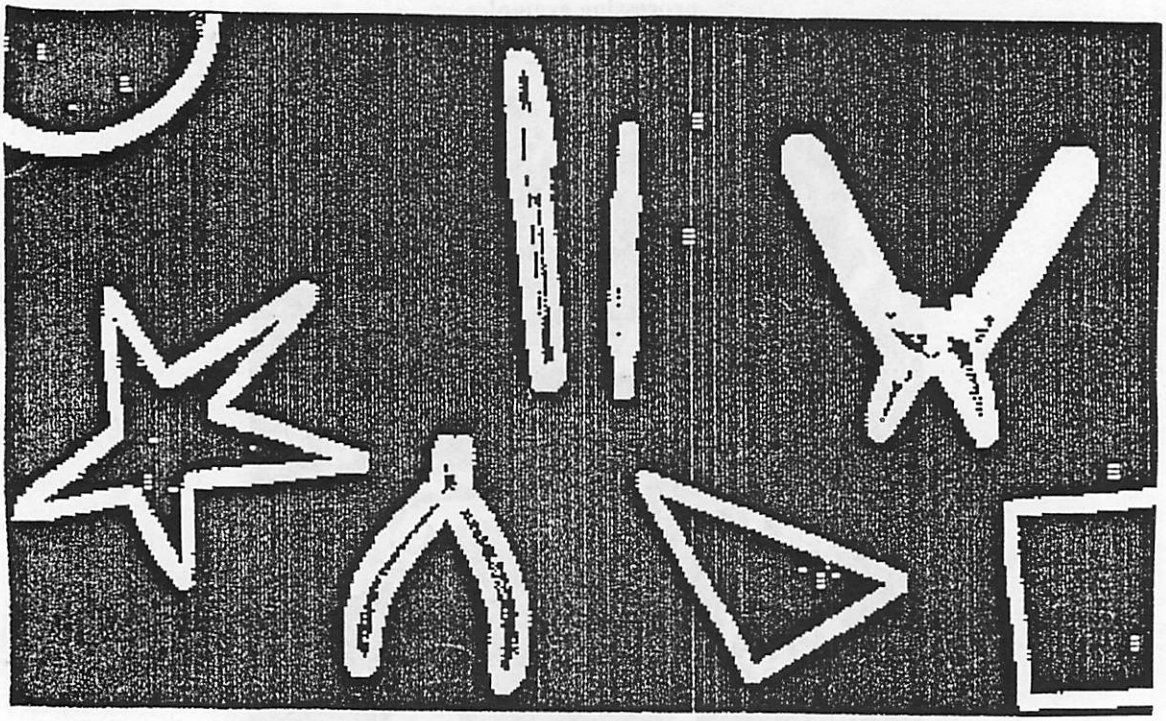


Figure 10b. Edges Extracted, Sobel Filter

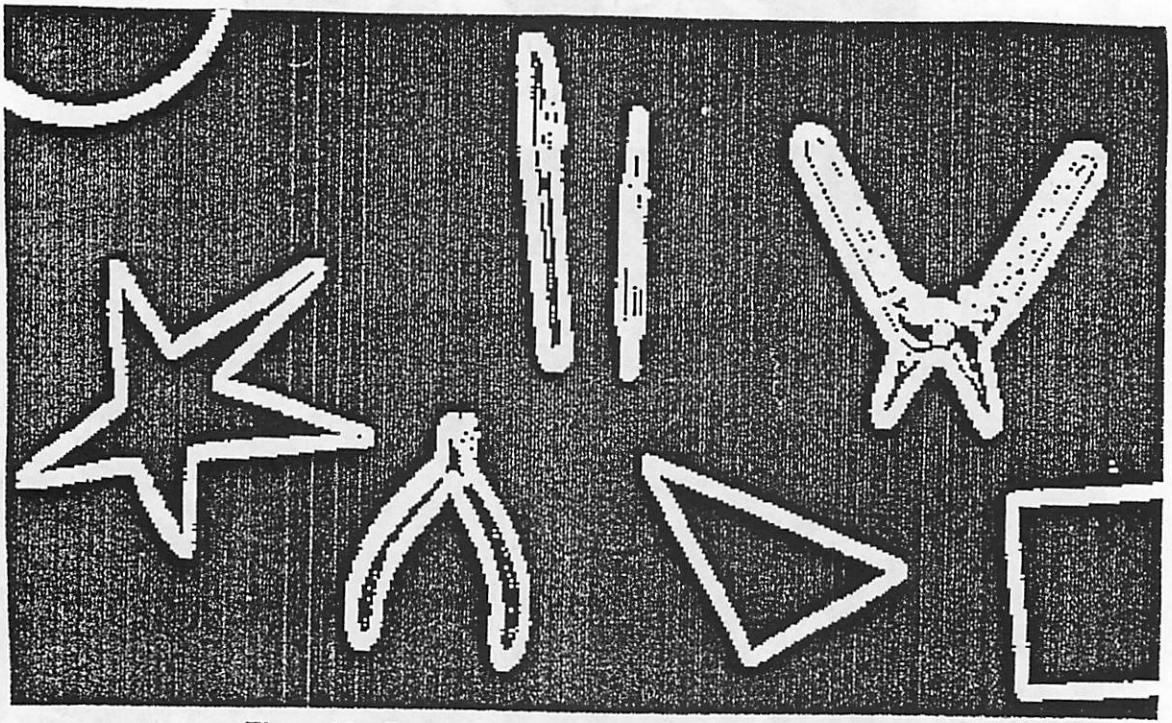


Figure 10c. Edges Extracted, Maximum-Difference Filter

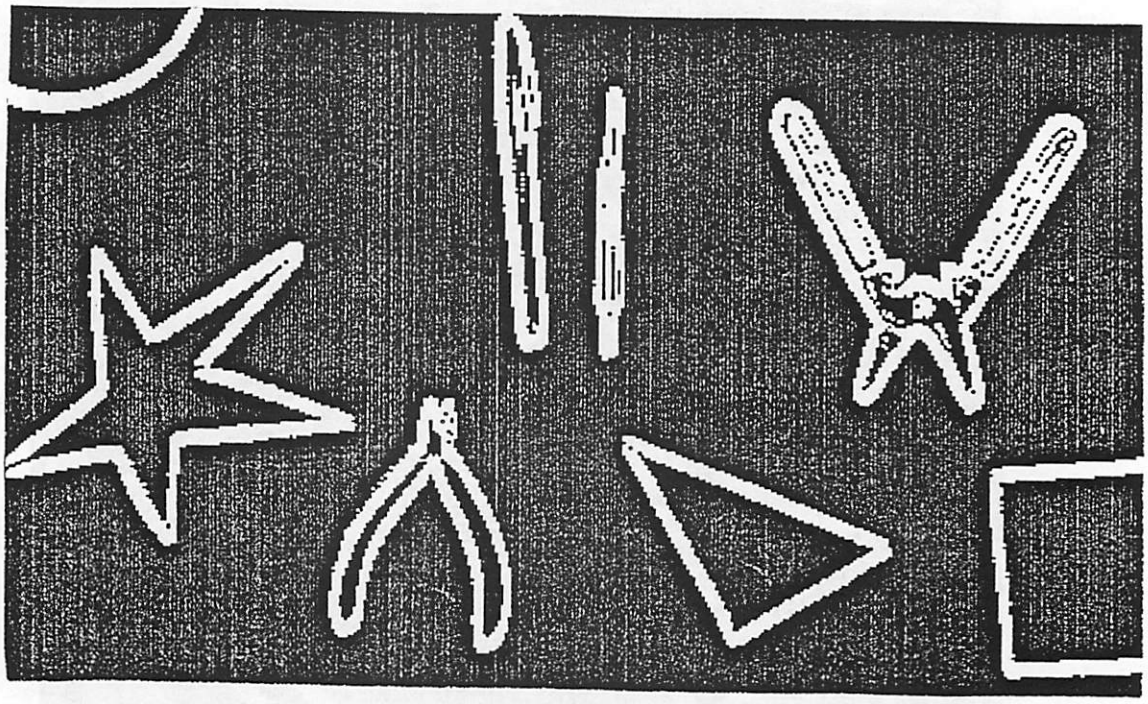


**Figure 10d. Edges Extracted, Compass Gradient Filter**

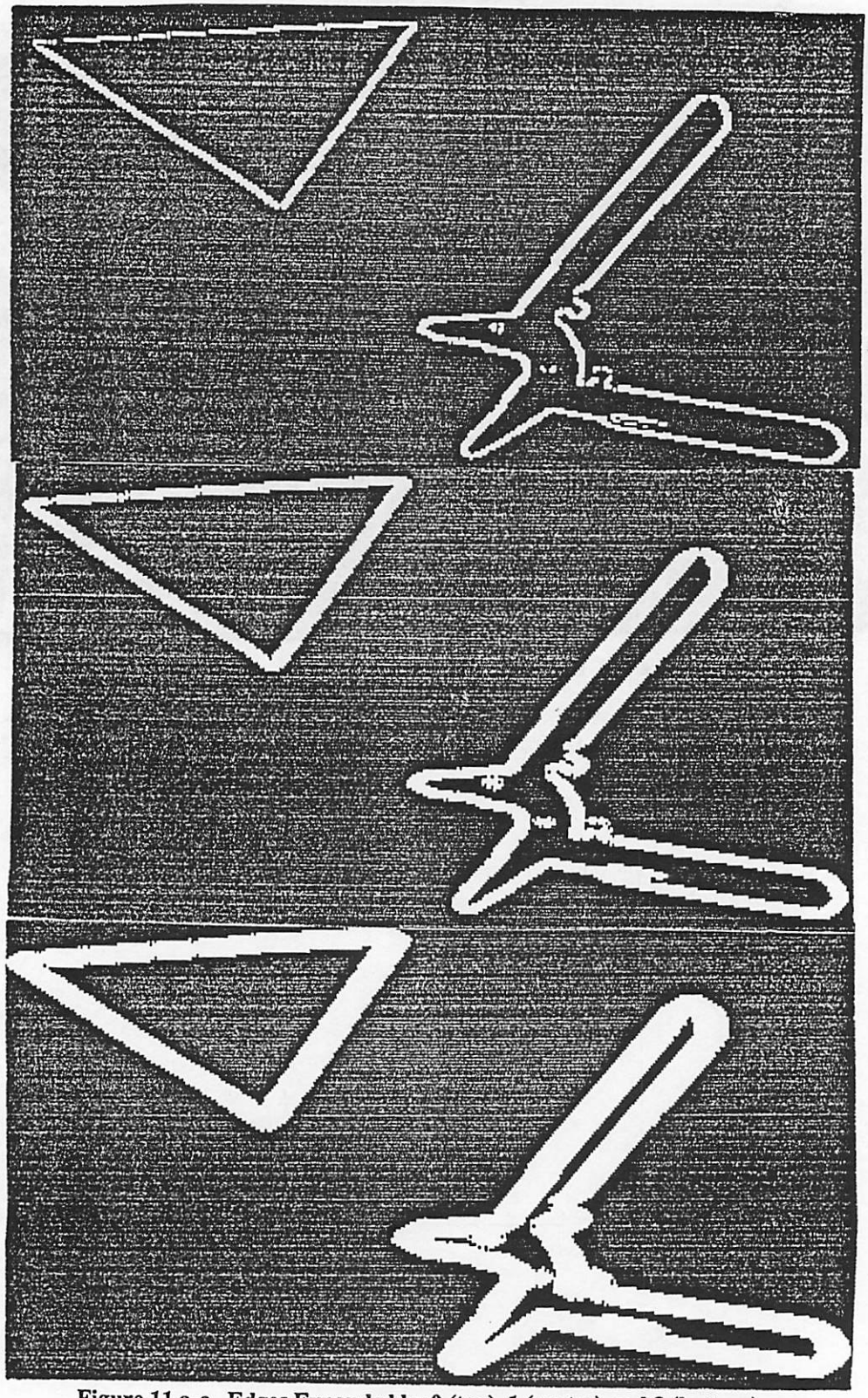


**Figure 11 a-c. Edges Expanded by 0 (top), 1 (center) and 3 (bottom)**

## 2.9 REFERENCES

[1] Gonzalez R. C., Wintz P. W., *Digital Image Processing*, Addison-Wesley, Reading, Mass., pp. 118-119.

[2] Prewitt J. M. S., Mendelsohn M. L., "The Analysis of Cell Images," *Ann. N. Y. Acad. Sci.*, 128, 1966, pp. 259-267.

[1] Gonzalez R. C., Wintz P. W., *Digital Image Processing*, Addison-Wesley, Reading, Mass., pp. 115-117.

[4] Andrews H. C., et. al., "Image Processing by Digital Computer," *IEEE Spectrum*, 9, 7, July 1972, 20-32.

[5] Hall E. L., "Almost Uniform Distribution for Computer Image Enhancement," *IEEE Trans. Computers*, C-23, 2, Feb. 1974, 207-208.

[6] Oppenheim A. V., Schafer R. W., Stockham T. G., Jr., "Nonlinear Filtering of Multiplied and Convolved Signals," *Proc. IEEE*, 56, 8, Aug. 1968, 1264-1291.

[7] Pratt W. K., *Digital Image Processing*, John Wiley & Sons, 1978, pp. 320-323, 479-485.

[8] Pratt W. K., *Digital Image Processing*, John Wiley & Sons, 1978, pp. 320-323.

[9] Graham R. E., "Snow-Removal: A Noise-Stripping Process for Picture Signals," *IRE Trans. Inf. Theory*, IT-8, 1, Feb. 1962, 129-144.

[10] Tukey J. W., *Exploratory Data Analysis*, Addison-Wesley, Reading, Mass. 1971.

[11] Healy G., Sanz J. L. C., "CONTAM: An Edge-Based Approach to Segmenting Images with Irregular Objects," *IBM Research Report*, Jan. 1985.

[12] Pratt W. K., *Digital Image Processing*, John Wiley & Sons, 1978, pp. 482.

[13] Pratt W. K., *Digital Image Processing*, John Wiley & Sons, 1978, pp. 495-499.

[14] Duda R. O., Hart P. E., *Pattern Classification* and *Scene Analysis*, Wiley, New York, 1973.

[15] Prewitt J. M. S., "Object Enhancement and Extraction," *Picture Processing* and *Psychopictorics*, Academic Press, New York, 1970.

[16] Rosenfeld, "A Nonlinear Edge Detection Technique," *Proc. IEEE Letters*, 58, 5, May 1970, 814-816.

[17] Nalwa, V. S., "On Detecting Edges," *Proc. Image Understanding Workshop*, Oct. 1984, pp. 157-164.

[18] Mandeville J. R., "Novel Method for Analysis of Printed Circuit Images," *IBM Journal of Research* and *Development*, Jan. 1985, pp. 73-86.

[19] McCubbrey, D. L., Lougheed, R. M., "Morphological Image Analysis Using a Raster Pipeline Processor," *Computer Architecture* for *Pattern Analysis* and *Image Database Management*, Nov. 1985, pp. 444-451.

[20] Pratt W. K., *Digital Image Processing*, John Wiley & Sons, 1978, pp. 502-503.

[21] Ballard, D. H., Brown, C. M., *Computer Vision*, Prentice Hall, 1982, pp. 85-88.

# CHAPTER III IMAGE RECOGNITION TECHNIQUES

## 3.1 INTRODUCTION

There is, at present, considerable effort going on in the field of image recognition and scene analysis [6-11]. At one end of the spectrum there is the general image recognition efforts in which few assumptions are utilized about the images of interest and the scenes which contain them. On the other end are the more practically oriented schemes which make use of known information to simplify the problem. As more specific information regarding the images is used, the recognition system becomes less general. However, the computational requirements should decrease and the performance of the system should improve as the range of applications is restricted.

An image recognition system was developed to recognize a class of images. Currently, the class is restricted to 2 dimensional objects which are characterized by their closed outer contours. This may seem to be a severe restriction, but often the images of interest are located in a known plane in space. In industrial assembly lines, some parts are transported on conveyer belts which are in a plane. Robots could use the recognition results to sort parts by type or remove defective parts in an inspection system. By restricting the class of images that can be recognized to those that are characterized by closed contours, some types of problems can not be solved. For example, inspecting some characteristic of a part that is inside the outer contour is not possible with this recognition system.

To make the recognition process more general, the decision should be insensitive to the rotation, size and the absolute position of the object. In some systems, insensitivity to rotation and or size may not be required and actually may be undesirable . For example in an assembly line application, if the camera remains at a fixed distance from the image plane, the size invariance would not be needed since the same object will always appear to be the same size. Size invariance would be undesirable if it is desired to

recognize two objects with the same shape but different sizes. For these cases, it is often possible to utilize more information and improve the performance of the recognizer with some reduction in the generality of the system.

In addition to the identity of the object, it is interesting to know the position, orientation and size of the object. This information makes it possible to perform some action on the object after it has been identified. The principle axis of the object may also be of interest if a robot "hand" is to pick up the object.

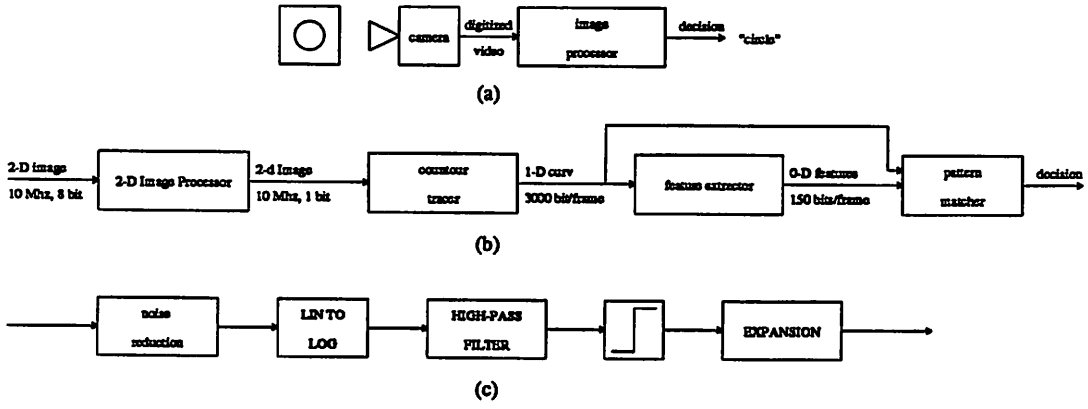



**Figure 1. The Image Recognition System**

The basic set-up is illustrated in figure 1a. The real-time video image from a camera is digitized and sent to the image processing hardware. The processing hardware generates a low rate decision and a description of the location of the object which can be used by the outside world.

Some of the techniques that are available for performing this type of image recognition and the systems that are needed to generate the desired features will be discussed in this chapter.

## 3.2 RAW DATA TO BE UTILIZED

A contour tracer is the primary source of information for the recognizer to work with. The tracer follows along the outside edge of the image and outputs the curvature of the contour at every pixel. The absolute coordinates and the direction between pixels are also available. The curvature was chosen as the primary output signal because it can be represented in 3 bits (the coordinates require 14 bits) and the cur-

vature is invariant with respect to translation, rotation and size of the image if it is scaled and shifted appropriately. It is important to note that the contour tracer does not discard any information about the contour. From the curvature signal and initial conditions on the position and the direction, the entire contour can be unambiguously reconstructed. This allows many different recognition schemes to utilize the tracer output, because no assumptions regarding the use of the data has been built into the tracer except that only the object's outer boundary is of significance.

The curvature computed is really the discrete curvature. The continuous curvature and the discrete curvature are defined below. $\phi(s)$ is the angle of the tangent at each point on the continuous contour and $\phi(n)$ is the angle between adjacent points on the discrete contour.

$$curvature\,(s) = \frac{d\phi(s)}{ds} \quad where \quad \phi(s) = \tan^{-1}\left(\frac{dx(s)/ds}{dy(s)/ds}\right)$$

and *the contour is represented by* $(x(s),y(s))$

$$discrete\ curvature\,(n) = \frac{\Delta\phi(n)}{\Delta n} \quad where \quad \phi(n) = \tan^{-1}\left(\frac{\Delta x(n)}{\Delta y(n)}\right)$$

and *the contour is represented by the sequence*, $(x(n),y(n))$

$$\Delta x(n) = x(n)-x(n-1),\ \Delta y(n) = y(n)-y(n-1),\ \Delta n = n-(n-1)=1$$

The curvature basically indicates how fast the contour is changing. Near sharp corners, the curvature has a large value while the curvature is zero for straight lines. With the continuous curvature, the problem arises that it is unbounded at discrete corners, for example, the corners of a square. The discrete curvature is always bounded because $\Delta n$ is always 1. By matching the curvature signals, one is essentially matching the "corners" or abrupt changes in the contour. Intuitively, this seems to be good thing, since the human visual systems seems to utilize the corners of an image to a large extent.

There is one fundamental problem with using the discrete curvature. Because $\Delta x$ and $\Delta y$ can only have the values -1,0 or 1, $\phi$ will always be $n\frac{\pi}{4}$ (only 8 distinct values) and the curvature will effectively be quantized to only three bits. Another way to see this is that each pixel has only 8 adjacent neighbors and hence $\phi$ and the curvature can only have 8 distinct values.

The three bit quantization of the curvature results in a very low SNR which can be seen in figure 2a which shows the raw curvature signal for a 5 pointed star. This is the signal from which the entire contour can be reconstructed. The background quantization noise is evident and a few small spikes can be

seen which correspond to the sharp corners of the star. The effect of filtering on the curvature signal can be seen in figures 2b-d. The simple filters used in figures 2b-d are given in table 1. As the filter cutoff frequency is reduced (from figure 2b to 2d), the quantization noise and fine features in the curvature signal disappear while the major features in the curvature signal, that is, the 5 points and 5 corners between points, remain.

To get useful information from the curvature signal other than the actual (X,Y) coordinates of the contour, some low-pass filtering appears to be necessary. Actually, the coordinates are obtained by integrating the curvature twice (i.e. low-pass filtering). The problem with filtering the curvature signal is that the filtered curvature varies with image scaling and size invariance is lost. The spectrum of the curvature signal spreads as the image is compressed according to the relationship:

$$\text{if } curv(n) \rightarrow H(\omega)$$
$$\text{then } curv(fn) \rightarrow \frac{1}{f} H\left(\frac{\omega}{f}\right)$$

As the image is scaled down, features will be filtered out that would pass through the low-pass filter when the image is larger. To solve this problem, either the filter must be adapted to the image size or the image must always be large enough to ensure that important features will not be filtered out while the quantization noise will be attenuated sufficiently. Basically, it is required that the image be oversampled to make it possible to obtain a larger SNR than that obtained by the original quantization.

| figure | $H(z)$ |
|---|---|
| 2b | $\frac{\sum_{n=0}^{3} z^{-n}}{1-\frac{3}{4}z^{-1}}$ |
| 2c | $\frac{\sum_{m=0}^{3} z^{-m} \sum_{n=0}^{3} z^{-n}}{1-\frac{3}{4}z^{-1}}$ |
| 2d | $\frac{\sum_{m=0}^{10} z^{-m} \sum_{n=0}^{3} z^{-n}}{1-\frac{3}{4}z^{-1}}$ |

**Table 1. Filter Transfer Functions used in Figures 2b-d**

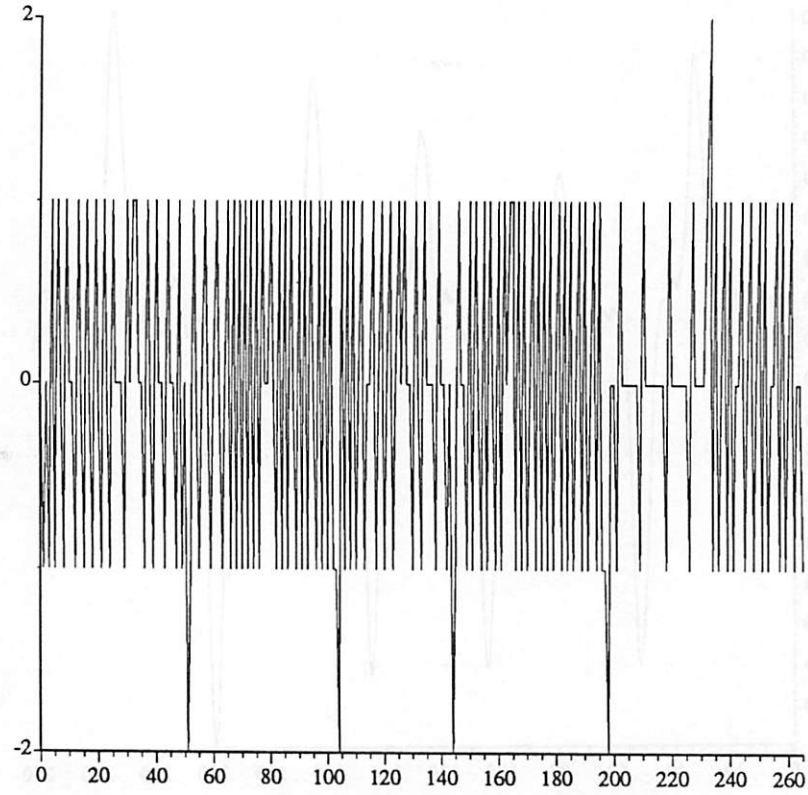



**Figure 2a. Original Curvature for 5 Pointed Star**

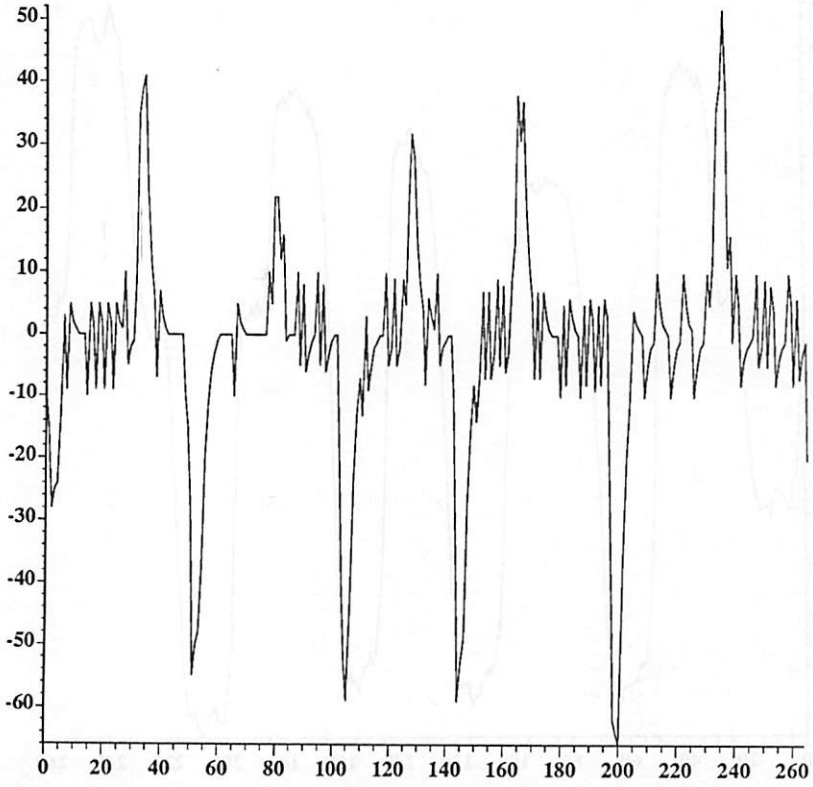



**Figure 2b. Filtered Curvature for 5 Pointed Star**

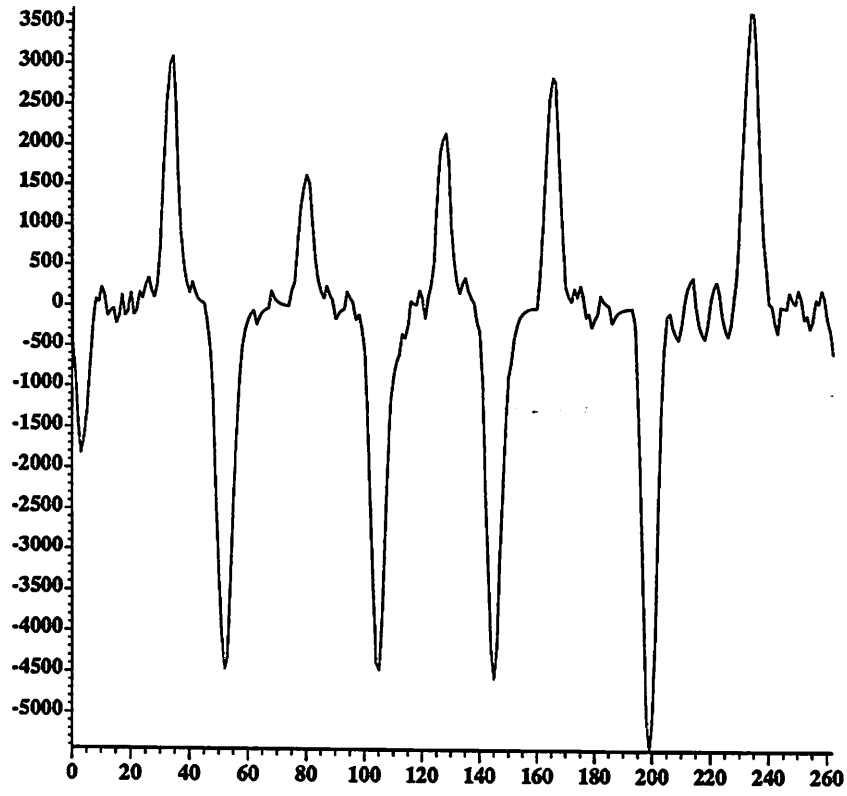



**Figure 2c. Filtered Curvature for 5 Pointed Star**

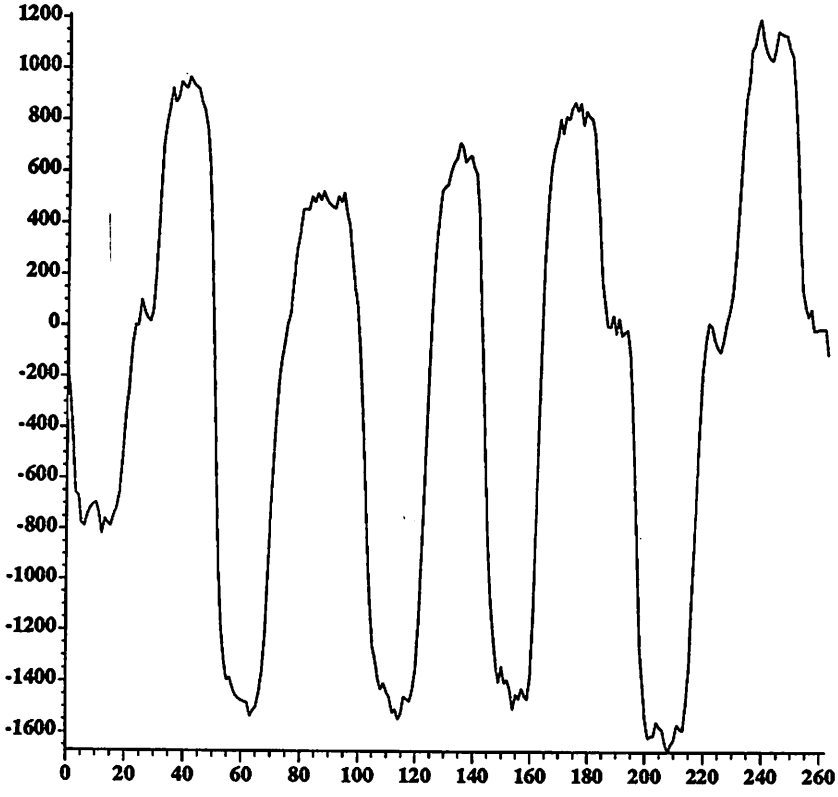



**Figure 2d. Filtered Curvature for 5 Pointed Star**

## 3.3 THE SYSTEM

The complete recognition system is shown in figure 1. Figure 1b shows a high level representation of the system. The 10 Mhz video is processed by the 2-D image processor which outputs an edge map image ready for contour tracing. The contour tracer reduces the number of bits required to represent the image by generating the curvature signal. The feature extractor computes several features for a feature-based recognizer.

The 2-D image processor is shown in more detail in figure 1c. The 8-bit 10 MHz video signal is first processed by a noise filter to reduce the number of false edge points generated by noise. Currently, a median filter is used to reduce the noise. The "cleaned" video signal is then passed through a point-wise linear to logarithmic converter to reduce the sensitivity to illumination variations. The output of the log converter is high-pass filtered to enhance the edges. A Sobel filter is used with a sum of absolute values being performed instead of the square root of the sum of squares. The high-pass video image is then thresholded to generate the 1-bit edge map. The edge map is dilated or expanded by logical convolution to join broken contours and to reduce the bandwidth of the signal before down-sampling in the contour tracer. All blocks in the 2-D processor generate a 2-D raster representation of an image from an input 2-D raster representation.

Currently, all of the processing blocks in figure 1 (except the pattern matcher) have been implemented with custom chips. The pattern matching is being performed on a SUN work station. It would be possible to either develop a custom chip to perform this task or to use a fast general purpose signal processor. At present, very little processing is performed by the SUN and hence it does not significantly slow down the recognition process.

## 3.4 RECOGNITION FEATURES

There are many different ways to discriminate objects based upon the basic information that is output by the contour tracer. Basic features include Fourier descriptors [1], invariant moments[2] and the curvature signal itself. There are several other features that can be computed from the curvature of an object.

### 3.4.1 Fourier Descriptors

To generate the Fourier descriptors, the set of (X,Y) points of the contour is parameterized by a single variable. A complex sequence, $z_n$, is generated from the (X,Y) pair by setting $z_n = X_n + jY_n$. The positions on the contour represented in this way are a complex-valued periodic sequence of a single parameter. The DFT of $z_n$ can be obtained but the resulting DFT coefficients are not invariant with respect to translation, rotation and size. The DC term represents the position of the center of mass and can be neglected for recognition. If the image is rotated a phase shift appears in all terms that must be removed. Scaling of the image results in a scaling of the Fourier coefficients.

Different ways for normalizing the Fourier coefficients have been suggested from simply normalizing the magnitude of each coefficient by the magnitude of the first harmonic [3] to more complicated normalization schemes. In the first scheme, the normalized coefficients are invariant to translation, rotation and scaling because the term with absolute position information is discarded, the phase information which contains rotational information is also discarded and the rest of the coefficients are normalized to remove scaling information. Granlund [1] suggests a more complicated normalization technique which is supposed to retain more "significant" information regarding the shape of the image.

Tests have shown that the first 8 normalized coefficients are sufficient for recognizing several different images but more coefficients may be needed to distinguish objects that are very similar. A standard FFT should not be used because the number of points will, in general, not be a power of two. Further, it is not necessary to compute all DFT coefficients. If only 8 normalized coefficients are needed, approximately 19 coefficients must be computed. The extra coefficients represent the terms used to normalize the final 8 coefficients and the negative frequency terms which are not the complex conjugate of the positive frequency terms.

The DFT computation requires approximately $2Nm$ complex multiply adds for input signals with $N$ points and $m$ normalized Fourier coefficients. The matching requires very few operations since only $m$ (typically 8) numbers must be compared for each image.

### 3.4.2 Invariant Moments

Invariant moments have been used to characterize objects. The basic technique is to compute some number of moments for the object where the $p,q^{th}$ *moment* is defined by:

$$M_{p,q} = \sum_x \sum_y x^p y^q f_{x,y}$$

where $f_{x,y}$ is 1 if the point $(x,y)$ is inside the object and 0 otherwise. As with the Fourier descriptors, these basic moments are not invariant with respect to translation, size or rotation. By normalizing the computed moments, a set of invariant moments is obtained. For example, invariance to translation can be obtained by replacing $x^p$ with $(x-\bar{x})^p$ and $y^q$ with $(y-\bar{y})^q$ in the equation above. Again, it has been found that only a few moments are necessary to distinguish several objects.

### 3.4.3 Curvature Signal

The curvature can be matched directly to previously measured curvature signals for the template images if the sequences are scaled and shifted with respect to each other. Spatial scaling, making the sequences the same length, removes the effects of size variations and shifting accounts for a possible relative rotation. These one dimensional signals can in principle be treated just like speech signals and matched similarly [4].

When matching the curvature signals directly, the two signals must first be made the same length. When scaling, the choice must be made whether to simply decimate the longer length signal or interpolate the the shorter length signal or to use a dynamic time warp algorithm. Once the two signals are the same length, the difference between the two can be computed at each point and the square of the error accumulated. This accumulated error is called the score and must be computed for every possible alignment. The lowest score among the different orientations is taken to be the score between the signals. The process is then repeated for each test image. The current image is assumed to be the same as the template image that produced the lowest score.

If $N$ is the signal length and $k$ is the number of template images, the direct matching technique requires at most $kN^2$ subtract-square- accumulates in addition to the operations required for the size normalization and the filtering. This number can be reduced by terminating the matching process if the error

for the current alignment exceeds the minimum error computed for a previous alignment.

It can be shown that these operations can be performed in the frequency domain. The total number of operations can be reduced by using fast transforms to only $kNlog(N)$ if the sequences can be made to have lengths that are powers of two. Matching the auto-correlation coefficients instead of the full curvature has also been suggested. This would reduce the total number of computations to only $N^{2+kN}$, or less, if not all $N$ coefficients are needed in the matching procedure. Both of these techniques would require additional algorithm development.

### 3.4.4 Feature based recognition

A very promising recognition technique is based upon features which can be computed as the contour is traced. Currently, three features are being investigated: the ratio of area to perimeter squared [5] ($\frac{A}{P^2}$), the sum of positive values of the low-pass filtered curvature signal (FCS), and the variance of the FCS.

Essentially, $\frac{A}{P^2}$ measures the "circularity" of the image. The ratio has a maximum value for a circle and is zero for a line. The variance of the FCS reflects the "pointyness" of the image. Ideally, a circle has constant curvature and therefore zero variance of the FCS. For "pointy" objects, such as a multi-pointed star, the variance of the FCS is large. The sum of positive values of the FCS is a function of how far the image deviates from a convex shape. For all convex objects, the FCS always has a negative sign (and sums to a constant) and the sum of positive values is zero. For non-convex shapes the FCS can be positive at some points. The larger the sum of positive values of the FCS, the more areas of the image are concave.

Calculations and simulations showed that these features were reasonably distinct for some simple objects and hence could be used as the basis of a feature-based recognizer.

It is obvious that, in the continuous case, $\frac{A}{P^2}$ is invariant with respect to translation and rotation of the object from the definitions of area and perimeter. Since the area and the perimeter squared have units of linear dimension squared, scaling does not effect the ratio. This can be shown more rigorously by the

examining the definitions of area and perimeter in polar coordinates:

$$P = \int r(\theta) d\theta$$

$$A = \int \frac{1}{2} r^2(\theta) d\theta$$

if the object is scaled by $k$ (such that $r_k(\theta) = k r_1(\theta)$):

$$P(k) = \int r_k(\theta) d\theta = \int k r_1(\theta) d\theta = kP(1)$$

$$A(k) = \int \frac{1}{2} r_k^2(\theta) d\theta = \int \frac{1}{2} k^2 r_1^2(\theta) d\theta = k^2 A(1)$$

then:

$$\frac{A(k)}{P^2(k)} = \frac{k^2 A(1)}{k^2 P^2(1)} = \frac{A(1)}{P^2(1)}$$

The continuous curvature signal will not vary with the rotation of the object except for a shift in the curvature. Therefore, ideally, the sum of positive values and the variance of the FCS will be invariant with respect to the object orientation.

Features based on the FCS are not invariant with respect to the object size. The problem can be seen by looking at two types of contours, one with continuous slope or bounded curvature (e.g. an ellipse) and one with discrete changes of slope or unbounded curvature (e.g. a square). It can be shown that the variance of the curvature for the first and second cases varies differently as the size is varied. If the contour is represented by $r_k = kr(\theta)$, where $k$ is a spatial scaling factor, then for the continuous curvature case, $\sigma^2_{curv}(k) = \frac{1}{k^2}\sigma^2_{curv}(1)$.

This relationship can be established by noting that the curvature in polar coordinates is given by:

$$curv(k) = \frac{r_k^2 + 2\left[\frac{dr_k}{d\theta}\right]^2 - r_k \frac{d^2 r_k}{d\theta^2}}{\left[r_k^2 + \left[\frac{dr_k}{d\theta}\right]^2\right]^{\frac{3}{2}}} = \frac{1}{k} curv(1)$$

$$then \quad \sigma^2_{curv}(k) = \frac{1}{k^2}\sigma^2_{curv}(1)$$

For the discrete curvature case, the curvature as a function of the scaling factor $k$ and the position on the contour, $n$, is:

$$curv_n(k) = \sum_{i=0}^{l} a_i \delta(n - kn_i)$$

and the variance is defined as:

$$\sigma^2_{curv}(k) = E_n[curv_n^2(k)] - E_n^2[curv_n(k)]$$

where the estimate of the expected value of $curv^2$ is given by:

$$E_n[curv_n^2(k)] = \frac{1}{P(k)} \sum_{n=0}^{kN_0} curv_n^2(k) = \frac{1}{kP(1)} \sum_{n=0}^{kN_0} \sum_{i=0}^{l} a_i^2 \delta(n - kn_i)$$

$$= \frac{1}{kP(1)} \sum_{i=0}^{l} a_i^2 \sum_{n=0}^{kN_0} \delta(n - kn_i) = \frac{1}{kP(1)} \sum_{i=0}^{l} a_i^2 = \frac{a}{k}$$

similarly, the estimate of the mean of the curvature is:

$$E_n^2[curv_n(k)] = \left[ \frac{1}{k} \sum_{i=0}^{l} a_i \right]^2 = \frac{b}{k^2}$$

so, the estimate of the variance is:

$$k\sigma^2_{curv}(k) = a - \frac{b}{k}$$

The difference between the two cases can be explained qualitatively. In the first case, the variance of curvature decreases as the image is scaled up because the curvature itself decreases in magnitude by $\frac{1}{k}$. In the second case, the curvature does not change in magnitude, but the contour is longer and hence the variance only decreases by $\frac{1}{k}$. If it is assumed that the image is dominated by sharp corners and the values of $a$ or $k$ are sufficiently large, then the value $k\sigma^2_{curv}(k)$ should be reasonably insensitive with respect to size variation.

Although $\sigma^2_{curv}$ is not invariant with respect to the object size, the feature is not totally useless. Because the relative size of the object can be easily determined, by computing the area or perimeter, features which vary with object size can be measured and stored for various object sizes. When recognizing, the size of the object can then be used to choose the most appropriate feature value for comparison from the set of stored values. This is not easily extended to the use of features that are not rotationally invariant, because there is no simple way of measuring the object orientation to choose the most appropriate feature value.

Although it is possible to use features that vary strongly with object size, it is convenient to use features which vary as little as possible. If the features do not vary greatly, it is possible to store only a

few values of the feature for different sizes and interpolate between the stored values.

## 3.5 REFERENCES

[1] Granlund G. H., "Fourier Preprocessing for Hand Print Character Recognition", *IEEE Tans. Comput*, vol. C-21, pp. 195-201, Feb. 1972.

[2] Hu M-K, "Visual Pattern Recognition by Moment Invariants", *IRE Trans. Iform. Theory*, vol IT-8, pp. 179-187, 1962.

[3] Gonzalez R. C., Wintz P., *Digital Image Processing*, Addison-Wesley, pp. 352, 1977.

[4] Kavaler R., et. al., "A Dynamic Time Warp IC for a One Thousand Word Recognition System", *Proc. ICASSP*, Vol. 2, pp. 25B.6.1-25B.6.4, Mar. 1984.

[5] Pratt W. K., *Digital Image Processing*, John Wiley and Sons, pp. 526, 1978.

[6] Petajan, E. D., "Automatic Lipreading to Enhance Speech Recognition," *Proc. CVPR* 85, June 1985, pp. 40-47.

[7] Skolnick, M. M., "Automatic Comparison of 2-D Electrophoretic Gels," *Proc. CVPR* 85, June 1985, pp. 48-54.

[8] Magee, M., "A Rule Based System for Pattern Recognition that Exploits Topological Constraints," *Proc. CVPR* 85, June 1985, pp. 62-67.

[9] Weymouth, T. E., et al., "Rule-Bases Strategies for Image Interpretation," *Proc. Image Understanding Workshop*, June 1983, pp. 193-202.

[10] Ketonen, J., "Deducing Facts about Scenes from Images," *Proc. Image Understanding Workshop*, June 1983, pp. 182-183.

[11] Barnard, S. T., et al., "Three-Dimensional Shape from Line Drawings," *Proc. Image Understanding Workshop*, June 1983, pp. 282-284.

# CHAPTER IV HARDWARE CHOICES

## 4.1 INTRODUCTION

Image processing algorithms have been implemented in many different types of hardware. These include: large arrays of processors (e.g. MPP [1,10], CLIP [2]), data-flow processors (e.g. ImPP [3]), pyramid connections of processors (e.g. NON-VON [4]), processors built from very high speed general purpose components and custom image processors that have architectures designed for a particular image processing algorithm. Within these broad groups is still the choice as to whether the data is processed in a bit-serial or bit-parallel manner. The choice of a particular type of hardware is dependent upon system tradeoffs, including: programming ease, cost, size, and expandability. To achieve real-time performance in a small area we have designed dedicated image processors with bit-parallel arithmetic (when the data is $n$ bits wide then we use a $n$ bit wide data path). With this choice, we can guarantee that all hardware elements are fully utilized and no excess hardware is included. However, the circuit design time (over all applications) is probably greater for the custom chips than for the arrays of identical processors.

## 4.2 MAJOR ISSUES

### 4.2.1 System Cost

Normally it is desired to minimize the "cost" of a particular system. The problem is that there are many types of costs. For example there is the development cost, which includes the cost of designing any ICs or other components which go into the system. There are also costs in building and operating the system. These include, materials (chips, boards, etc), power, maintenance and labor, which increases as the number of components in the system increases. Building compact, high performance processors minimizes the cost of building and operating the system while increasing the development cost. On the

other hand, large arrays of similar processors may decrease the development cost by increasing the other costs. If the development time of compact processors can be reduced, a lower total cost may be achievable. We have developed a design methodology which reduces the design time by re-using hardware in a hierarchical way and utilizing CAD tools to quickly assemble the remaining hardware.

### 4.2.2 Processors Constructed from High Speed MSI Components

One of the most common approaches to implementing image processing algorithms in hardware is to use fast bipolar adders and multipliers that are assembled at the system level from discrete chips. These systems usually have a very high cost because the components are expensive and require a large board area and consume a great deal of power (most of the fast adders and multipliers are bipolar circuits). Because the components are general purpose, pipelining (and hence lower power circuits) often can not be employed even when it is allowed by the algorithms. Further, little or no parallelism is obtained within each chip so many chips must be used to increase the system throughput.

Systems built with these components would not be suitable for high volume products which require low power consumption and small system size. They are most useful for low-volume programmable development systems that need higher processing speeds than those commonly found in general purpose computers.

### 4.2.3 Problems with General Purpose Image Processors

The problem with using general purpose processors, including arrays of similar (possibly systolic [9]) processors, data-flow machines and pyramid processors is that in general the hardware is not fully utilized and there is excess hardware overhead in the processor. The hardware will be under-utilized if some processing elements are inactive because data is not available due to I/O bandwidth problems and/or inadequate inter-processor communication or because the algorithm can simply be performed with fewer processors. Excess hardware includes control, computational and/or storage elements that are not inherently necessary for performing the algorithm. Processors which try to span a wide range of diverse applications will have to include all hardware that is necessary for any of the applications. Inevitablely, some of this hardware will not be needed for some algorithms. For example, a general purpose processor

may include an ALU that can perform multiplications, additions and logical operations. If only power of two multiplications are required, a fully parallel multiplier would be excess hardware. If only AND and OR operations were needed (as is the case in a class of image processing algorithms [chapter 2, 18 and 19]), the adder-multiplier section would be useless. Because of these problems, solutions using these types of processors typically require much more hardware for a given application than a solution utilizing custom circuits. If more hardware is used than is needed, the system cost will increase due to greater space and power demands. Further, because the hardware of these "general purpose" processors is fixed, the algorithm will often require more time or hardware for execution than it would in a custom architecture that is better suited to the particular application.

This problem is much more severe for image processing (with a sample rate that is comparable to the circuit clock rate) than for audio signal processing. For the audio case, general purpose processors are available that can perform a wide range of tasks on a single chip (e.g. TMS 320). The reason for this is that most audio signal processing algorithms are based upon a few standard operations (multiplication and addition) and the circuit clock rate is typically several orders of magnitude greater than the sample rate. Therefore, a single fast processor with a RAM, adder and multiplier can be designed in which the hardware can be almost fully utilized and little excess hardware is included. Because there are many clock cycles per sample (typically) under utilized processors do not add to the system cost as long as the algorithm does not require more cycles for execution than $\frac{clock\ rate}{sample\ rate}$. In addition, any excess hardware only occurs once.

For most image-processing algorithms, a great deal of parallelism in the hardware is required to achieve real-time operation. The optimum inter-connection of these processors is dependent upon the particular algorithm being implemented. The operations performed by the processor also vary from algorithm to algorithm. For this case, any excess hardware and under utilized processors caused by poor inter-processor communication both add significantly to the total hardware costs of the system. The excess hardware of the processor is now multiplied by the number of processors (as noted, some suggest the use of 32K processors). Under utilized processors increase the over all hardware cost by increasing the total number of processors required to increase the throughput of the less efficient processors.

Of course, there is one major difference between most of the general image processors and the custom processors. The general processors can usually be programmed to handle a wider range of tasks while the custom processors perform a limited class of applications. This may be a desirable feature for algorithm development, but in an actual system where cost is important and the algorithms are fixed, a few custom chips which have been optimized for a particular class of algorithms may be preferable to many boards full of general purpose chips. In addition, it is important to note that the general purpose chips were designed to execute a class of algorithms. For algorithms outside of this class of anticipated algorithms, the performance of the general purpose hardware may decrease dramatically.

### 4.2.4 Bit Serial Vs. Bit Parallel

For any of the general classes of hardware discussed, it is possible to implement the arithmetic units in a bit-serial or bit-parallel manner. Many of the large processing arrays (MPP, CLIP) use bit-serial techniques. It is thought that since a very large array of processors is being created it is important to make each processor as small as possible. However, bit-serial arithmetic units are inherently less efficient in the utilization of silicon area than bit-parallel units when the sample rate and clock rate are the same (or nearly so). This is a result of the fact that state of the system must be stored in either case. For example, in an accumulator, the entire data word must be stored. In the bit serial case, there will be one bit of adder for every $N$ (the word width) bits of storage, while in the bit-parallel case there will be $N$ bits of adder for $N$ bits of storage. In the bit-serial scheme there must be $N$ times as many processors to maintain the same throughput. Under this condition, the number of single bit adder cells will be the same for both cases but the bit-serial hardware will have $N$ times the number of storage cells. Bit-serial processors also have the disadvantage that the parallel input pixel values must be converted to serial and the serial processor outputs must be converted to parallel. It seems that the goal of these processing arrays is to achieve a high degree of parallelism and bit-parallel arithmetic would be one way to achieve greater parallelism.

These ideas are illustrated in figures 1 and 2. Figure 1 shows a bit-serial accumulator with one adder bit and an $N$-bit shift register (the accumulator). In figure 2 a bit-parallel accumulator is shown. Now the $N$-bit shift register has been replaced by an $N$-bit parallel register and the single adder bit has

been replaced by a bit-parallel adder. If a system with $N$ bit-serial processors is used to achieve the same throughput as the bit-parallel processor, the bit-serial system will have over $N^2$ bits of storage compared to $N$ bits of storage for the bit-parallel system.

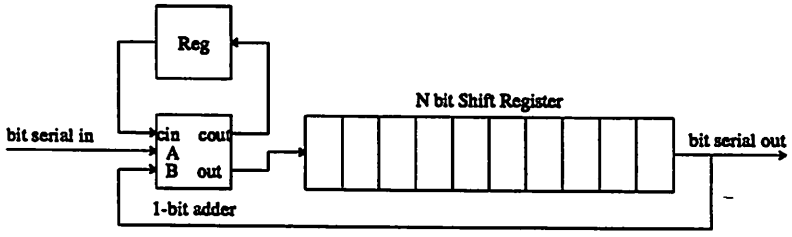



**Figure 1. Bit-Serial Accumulator**

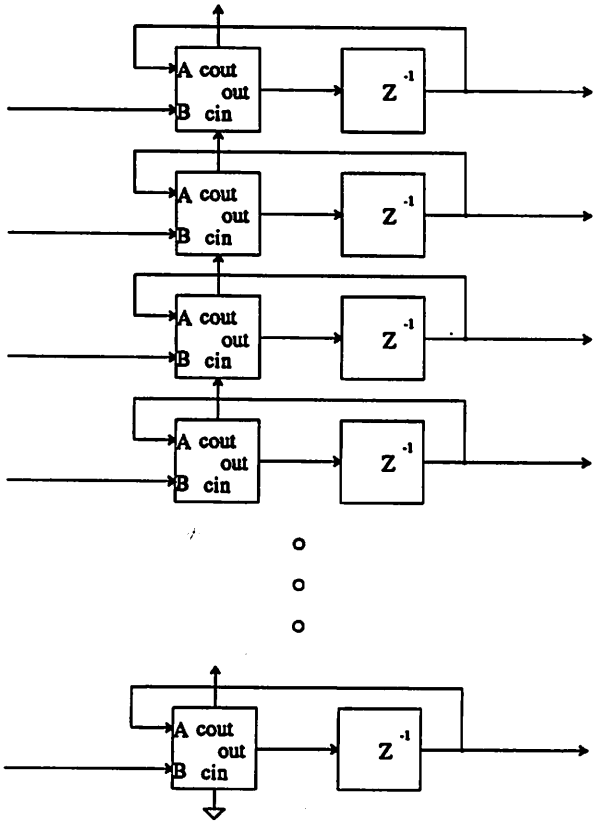



**Figure 2. Bit-Parallel Accumulator**

For audio band processing, this problem does not exist because the sample rate is much lower than the clock rate and only one adder bit needs to be included for each addition in the algorithm. The total

number of storage bits would be roughly the same for both the bit-serial and bit-parallel schemes.

## 4.3 COMPARISONS OF IMAGE PROCESSOR PERFORMANCE

Some comparisons of estimated execution times for different image processing algorithms on various image processors is shown in tables 1 and 2. It should be noted that all of the processors except the custom processors require external buffers to store the image frame. The relative strengths and weaknesses of each processor can be explained by examining the architecture of the processors.

| processor | # chips | external storage | function | | | |
|---|---|---|---|---|---|---|
| | | | median | thresh | 3x3 conv | 7x7 log conv |
| CLIP | many | yes | x | 4.5 | 45.5 | 1.74 |
| MPP | 2000 | yes | .28 | .04 | .69 | .24 |
| ImPP | 1 | yes | x | 66 | 3000 | 15000 |
| RISP | 1 | yes | x | 5.1 | 47.2 | 257 |
| Kung 1 | 1 | yes | N/A | N/A | 58 | N/A |
| Kung 2 | 1 | yes | N/A | N/A | 33 | N/A |
| custom 1 | 1 | no | 33 | N/A | N/A | N/A |
| custom 2 | 1 | no | N/A | 33 | N/A | N/A |
| custom 3 | 1 | no | N/A | N/A | 33 | N/A |
| custom 4 | 1 | no | N/A | N/A | N/A | 33 |

**Table 1. Image Processor Execution Time (in mSec) Comparisons for 512 x 512 Images**

| processor | function | | | |
|---|---|---|---|---|
| | area | perimeter | center of mass | total |
| CLIP | 4.0 | 0.5 | 3.0 | 36.0 |
| MPP | 0.5 | 0.5 | 3.0 | 4.0 |
| custom 5 & 6 | - | - | - | 24.0 |

**Table 2. Image Processor Execution Time (in mSec) Comparisons for 128 x 128 Images**

### 4.3.1 Single Chip General Purpose Processors

The ImPP data-flow machine exemplifies (figure 3) the problems of general purpose processors. This circuit has a general purpose arithmetic logic unit and a circular pipeline to prevent the system from becoming memory bound. The programmable controller reads the "tokens" attached to the incoming data

and decides if it should be queued for processing or passed through. When all pieces of data for a particular operation have arrived, the controller sends the data to the arithmetic unit for processing.

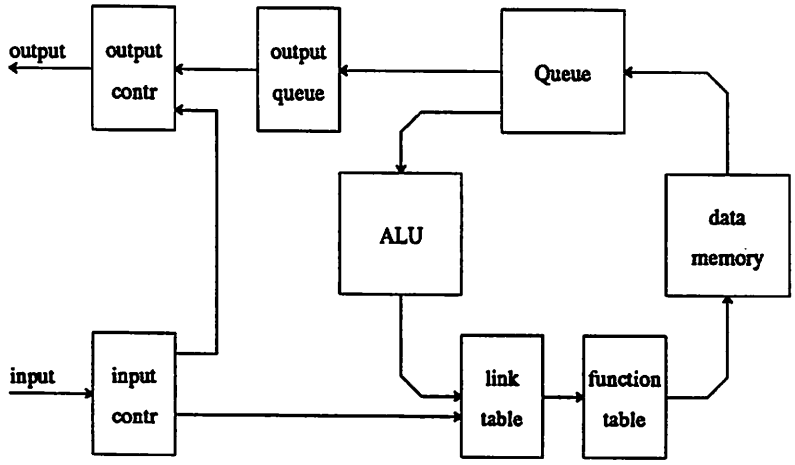



**Figure 3. ImPP Processor**

To perform these operations a fairly complex controller is required. Because the control circuitry is large, less room is available for computational elements. The data storage and arithmetic unit (a single alu and multiplier) occupy only about 35% of the chip area. Further, the multiplier and the adder must be used in different cycles so that arithmetic operations are only performed at a 5 Mhz rate. A common image processing task, 3x3 convolution, can be performed with this circuit. In real-time video signals, the data appears in a known order and the data-flow aspects of the chip are not necessary. What are needed are arithmetic units to perform at a 10 MHz rate the 9 multiplies and 8 additions (170 Mops/Sec) that are required to achieve real-time operation. Because resources are put into the control section instead of the arithmetic section of the chip, this chip takes 3 sec to perform the 3x3 convolution. The custom processor described in section 7.3 completes a 3x3 convolution in real-time (33 mSec). However, ImPP can be programmed to perform a wide range of tasks.

The two orders of magnitude reduction in throughput is caused partially by the use of a lower execution rate (5 MHz) for the ImPP (x2 reduction) and only being allowed to perform one arithmetic operation per cycle instead of 17 (x17 reduction). This only accounts for a throughput reduction by a factor of 34. The remaining degradation in throughput can probably be attributed to the need to fetch coefficients

for the multiplication operations from the data RAM and other general operational overhead.

ImPP is also much larger (in normalized units) than the custom processor. The ImPP processor was implemented in a 1.75 μ NMOS processor and was 7 mm on a side. This corresponds to area of 256 $mm^2$ when scaled for a 4 μ technology. The area of the custom processor, fabricated with a 4 μ technology, is only 42 $mm^2$.

It is also interesting to note that ImPP can not perform the 7x7 logical convolution faster than 3x3 linear convolution. This is because the bit-parallel architecture does not make it possible to perform single bit operations any faster and hence it takes longer to perform the operations on the 49 single-bit values in a 7x7 region than on the 9 multi-bit values in a 3x3 region. When performing logical convolution, the entire 16x16 multiplier is basically useless.

Another issue of importance is the time required for circuit development. The ImPP is a very complex (110K transistors) circuit that inevitably required a large development time. Our 3x3 convolver utilizes information about the convolution algorithm to allow the use a relatively simple array of processors that could be designed quickly.

The author of a paper describing a general purpose image processor [8] that contains more transistors than ImPP but does not have the complex data flow mechanisms, said that the total development time for the chip was 50 *man years*. In comparison, the total development time for our set of eight chips, including the 3x3 convolver, was only 1.5 *man years*.

Another single chip processor has been recently introduced for Real-time Image Signal Processing (RISP [5]). It is similar to ImPP except that RISP is not a data-flow processor and achieves a higher clock rate (cycle time is less than 20 *nSec*) by using a bipolar technology. In addition, with RISP it is possible to perform both a multiplication or division and an addition in the same cycle. The RISP processor (figure 4) has been designed to perform local image processing and stores a local region (5 x 5) of the image in a set of internal registers. In general, RISP looks very much like general purpose signal processors. It has a small data RAM (16 words), coefficient RAM (32 words) and a multiplier, accumulator structure. The processor is micro programmable.

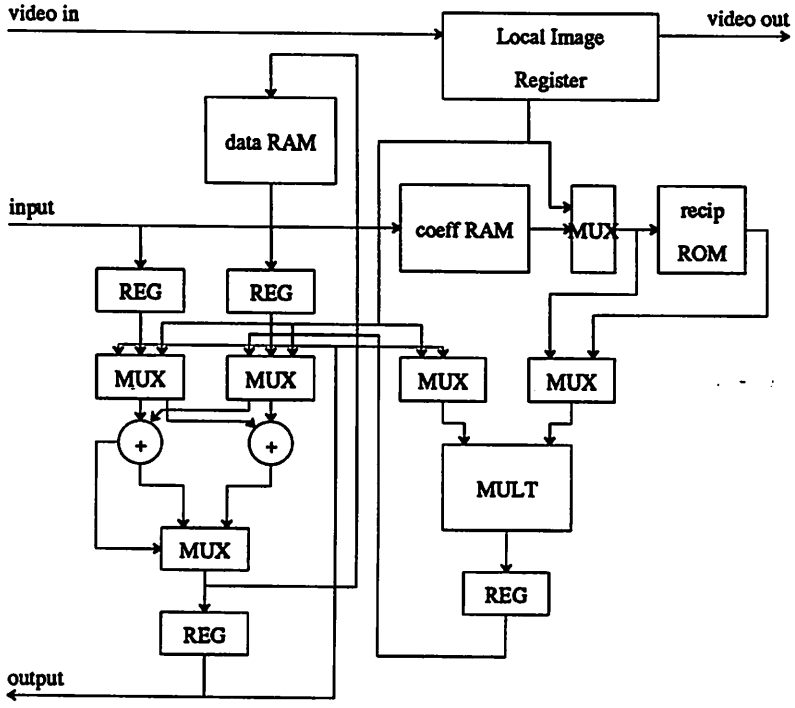



**Figure 4. RISP Processor**

It can be seen that this processor out performs the ImPP for the algorithms shown. However, this processor still can not perform the 3x3 linear convolution or the 7x7 logical convolution in real time. Although the circuit gets a speed advantage (approx. x5) over the custom processors by running at a higher clock rate, it loses this advantage by not having more parallel arithmetic units ($x\frac{17}{2}$ disadvantage for 3x3 convolution and $x\frac{49}{2}$ disadvantage for 7x7 convolution). The overall execution time for 3x3 convolution is about 1.7 times greater than that of the custom chip.

Although these general purpose single chip processors can perform local operations reasonably well, it is not clear how efficiently they could perform more global operations such as contour tracing, and the determination of the area, perimeter and center of mass of an object in the image.

### 4.3.2 General Purpose Array Processors

Arrays (figure 5) of thousands of processors (CLIP, MPP, NON-VON) suffer for similar though less dramatic reasons. Papers discussing the relative prowess of these machines usually include tables

showing orders of magnitude reductions in execution time for certain algorithms compared to a standard machine such as a VAX. The problem is that the execution time per processor is normally much higher for the large arrays than for the VAX. This indicates that the value (number of operations performed per unit time) per processor is much lower because the hardware is not fully utilized or because the architecture is poorly suited to the algorithm. One example, that results in poor processor utilization, is the computation of an image histogram.

Computing the histogram of an image (discussed in chapter II) simply requires the number of occurrences of each gray level be computed. This can be accomplished by storing the number of occurrences of each gray level (for typically 256 gray levels) in a RAM. For each pixel the value representing the previous number of occurrences of that gray level is incremented. This is repeated for each pixel in the image. It can be seen that only two memory accesses (one read and one write) and one increment need be performed per pixel. A custom processor consisting of a 256x18 RAM and one adder (with associated pipeline registers, etc) could perform this task in real time.

The 32K processor NON-VON machine is organized in a tree of planes. Each plane has half as many processors as the plane below it. The processors can communicate with 4 neighbors in the same plane. In addition, the processors in one plane can communicate with the parent in the plane above (each parent has two children).

Each processor has a bit-serial ALU with local RAM, registers and I/O circuitry and has a structure like that shown in figure 6 (MPP and CLIP also have a very similar structure).

The NON-VON machine barely achieves real-time (for a 128 x 128 image) performance when computing a histogram, requiring 4400 $\frac{processor\ \mu Sec}{increment}$. This means that each processor performed on the average only 1 increment (only required increments are counted) every 4400 $\mu Sec$. This shows that the processors are not performing useful operations very often. The 16K processor MPP performed much worse requiring 64 times real-time to compute the histogram or 128000 $\frac{processor\ \mu Sec}{increment}$. Obviously, these architectures are not well suited to this algorithm because a simple processor could easily achieve $1\frac{processor\ \mu Sec}{increment}$. A square array or pyramid array of processors is not well suited to comput-

ing histograms because of the overhead involved in having the individual processors each compute a part of the result and then combine the partial results.

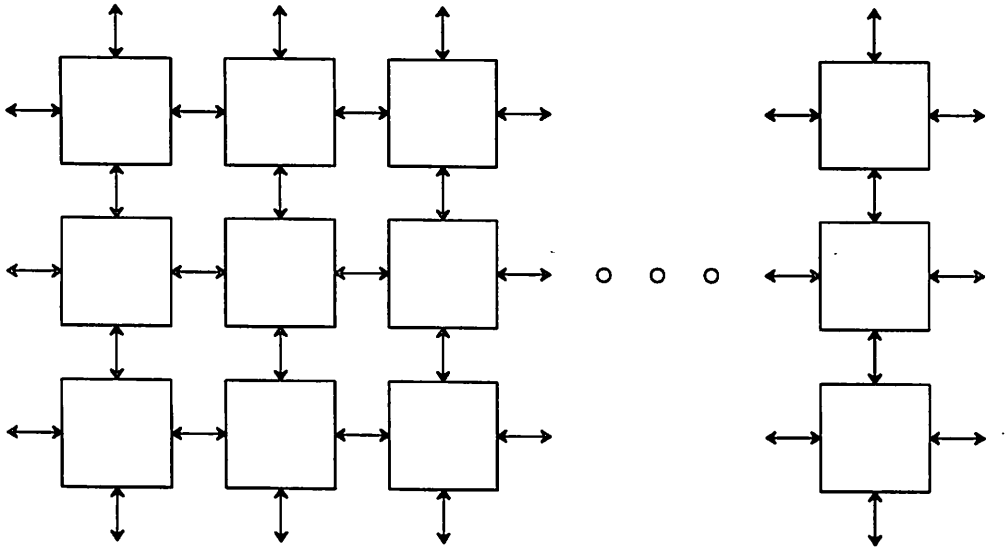


**Figure 5. Processor Array**

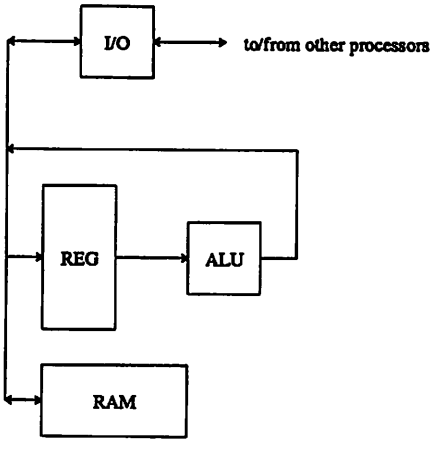



**Figure 6. Processor**

These array processors can typically perform local deterministic operations, such as convolution, fairly efficiently. The arrays are designed to communicate only locally which is sufficient for these algorithms.

The CLIP processor has an array of bit-serial processors arranged in a 96x96 array. Each processor has a limited amount of RAM and can communicate with each of its 8 nearest neighbors. The basic processor has a very simple arithmetic unit that can only perform logical operations.

The CLIP processor performs reasonably well with logical operations (e.g. logical convolution) but performs poorly with multi-bit operations. Each processor is not only bit-serial but does not even have a full adder cell. Therefore, 1-bit additions must be performed over several cycles so that multi-bit additions are much slower than single-bit operations. It should be noted that the 9.6K processor array performs 3x3 convolution (actually just a 3x3 sum) slower than a single chip implementation (custom 2). In addition, CLIP computes the area, perimeter and center of mass in more time than a two chip solution (custom 5 and 6). Although each arithmetic-logical unit should be quite small, each processor still has substantial I/O and storage (as discussed earlier in the comparisons between bit-serial and bit-parallel processors). Therefore, it appears that this approach results in a large system that performs worse than a single chip for some common algorithms.

It is interesting to see why an array of nearly 10K processors can not perform a 3x3 convolution in less time than a custom chip with 9 multiplying accumulators. First, CLIP has clock rate that is one fifth that of the custom chip. The bit-serial nature of CLIP reduces its throughput by a factor of $N = \textit{data word width} = 10$. Inefficient inter-processor communication reduces throughput by a factor of 2. Finally, because each processor does not have a full adder cell, the throughput is reduced further by a factor of about 20 (because each single bit addition requires about 20 cycles). Taking all of this into account, each CLIP processor has an effective throughput of only $\frac{1}{2000}$ of that of one custom processor so that the 9.6K processors provide the throughput of only 4.8 custom processors. The custom 3x3 convolver has 9 multiplying-accumulator processors and hence has greater processing power for performing 3x3 convolutions than CLIP.

When performing logical operations, CLIP performs better. Because logical operations are single-bit and do not require a full adder, the use of a bit-serial processor and the lack of a full adder does not degrade performance. In this case the effective throughput of each processor is only $\frac{1}{10}$ that of the cus-

tom processor. It can be seen from the table that the performance of CLIP compared to the custom processor is relatively better for logical than linear convolution.

The MPP was built by Goodyear for NASA. It has 16K processors organized in a plane with 128 processors on a side. Each processor has access to 1 Kbits of RAM also organized in planes (total RAM is 16 Mbit). The memory planes get data from and send data to the "S" plane. The "S" plane is a buffer that loads data from (or sends data to) the staging memory and transfers the data in one cycle to or from the RAM. The staging memory is another buffer which is used to convert data from a format used in the host computer to the plane format (bit serial) for use by the processors. In addition to a VAX 11/780 host computer, the system has a PDP-11 operating as a control processor.

Each processor contains a bit-serial adder, 6 one-bit registers, a variable length shift registers and control logic. Each processor can communicate with four neighbors.

MPP performs much better than CLIP and always achieves better throughput than the single chip solutions. The main reasons for this seem to be that MPP has a higher clock rate and better interprocessor and frame buffer I/O that seems to be able to keep the processors operating a large percentage of the time. In addition, there is a full adder in each MPP processor. MPP performs relatively worse when computing the area, perimeter and center of mass, indicating that the architecture is not as well suited to these types of computations.

MPP does require an enormous amount of hardware to achieve the high computation rate that it does. Eight MPP processors were put on a single chip, each 6.0x3.3 $mm^2$ in a 5 $\mu$ technology. A complete system is composed of 2000 of these chips which occupy 25300 $mm^2$ (normalized to a 4 $\mu$ process). For this system, the time-size product for 3x3 convolution on 512x512 images is 17.5 *Sec* $mm^2$ not including data storage and control hardware. For the custom 3x3 convolver chip, the time-size product is only 1.4 *Sec* $mm^2$ for the entire circuit. MPP also requires large amounts of storage (memory planes, "S" plane, the staging memory) and auxiliary control processors that were not taken into account in this measure.

### 4.3.3 What About I/O?

One thing that seems to be universally ignored when discussing these large computing arrays is where the data comes from and goes to. Presumably, in a real application there must be a source and sink of data. The amount of time taken to get data into and out of the array and whether this is a significant part of the total processing time is not discussed. For SIMD (single instruction multiple data streams) processors without additional hardware to quickly load all processors in the array with new data, it is likely that all processors will be inactive during the time when the data is being loaded into the array and read out of the array because the entire array will not have valid data. Without at least two complete storage arrays some frames could not be processed when operating from a standard video signal.

It is not obvious how the array performs when the number of processors in the array is smaller than the number of pixels in the image because, in this case, it may be necessary to load the image into the array part by part for processing. In this case, the I/O considerations are very important because it does not matter if the computation time can be reduced to 15 $\mu Sec$ if the loading of data takes 33 $mSec$.

### 4.3.4 Special Purpose Computing Arrays

Kung [6,7] has proposed two systolic arrays that perform specific functions such as 3x3 convolutions. The goal is to minimize the amount of excess hardware included in each processor and to keep all processors fully utilized by designing a processor dedicated to a single algorithm. This is very similar to the approach that we have chosen except that Kung has decided to keep the constraint of using systolic arrays.

The first attempt by Kung (Kung 1 in table 1) utilized a bit-serial basic processor (figure 7). A 3x3 array of 9 basic cells along with a row interface to sum the results from each row is used to compute a 3x3 convolution. However, because of the bit serial nature of the processor, the throughput is only $\frac{1}{8}$ that of a bit-parallel design. To increase the throughput, Kung suggests adding more (3) 3x3 arrays in parallel. This circuit was projected to produce only one valid result every 175 $nSec$. Disadvantages of this approach include the need for access to 5 lines of video data (for 3x3 convolution), the need for parallel to serial and serial to parallel converters and the need for many storage bits in the complete pro-

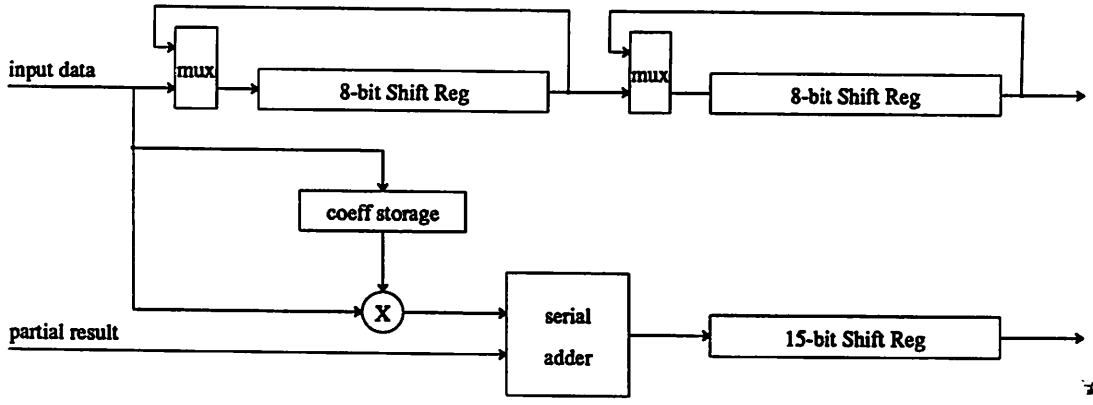



**Figure 7. Bit-Serial Systolic Convolution Cell**

cessor array. It can be seen that each processor has 32 bits of storage (registers needed to make the processor systolic). To achieve real-time operation, about 54 processors with a total of 1728 bits of storage (pipeline registers) will be needed. The custom chip has only 9 MACs, each of which has 3 10-bit registers for a total of 270 storage bits.

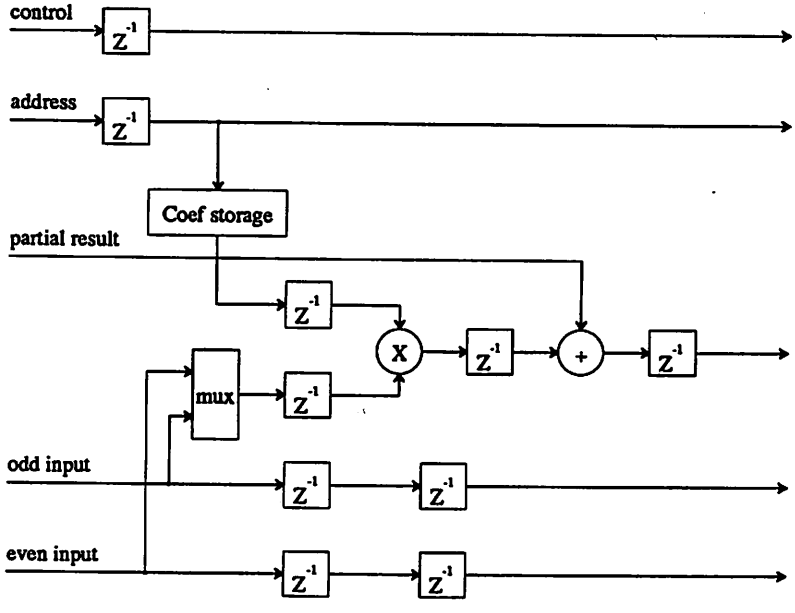



**Figure 8. Bit-Parallel Systolic Convolution Cell**

The basic processor that Kung proposes for a bit-parallel implementation is shown in figure 8. To make a *kxk* convolver out of these processors, $k^2$ basic processors are connected in a linear

configuration. A system made up of these processors requires only 2 streams of data for any length impulse response (with the previous processors, the number of data streams increased with the vertical length of the impulse response). Further, because he uses a one-dimensional array, the data in the two streams is required in an order that is different than that obtained by normal raster scanning of an image. This means that a random access frame buffer that can be read at twice the data rate would be required to generate the data for the systolic array.

Like the bit-parallel case, the bit-parallel processor has many registers that are required to make the processor systolic. Kung's convolver requires 90 registers, 9 adders and 9 multipliers while the custom circuit needs 27 registers, 12 adders and 9 multipliers. Because the registers are about 1/3 the size of a full adder, Kung's design would likely be larger if it were actually built.

## 4.4 DEDICATED ARCHITECTURES FOR SPACE EFFICIENT PROCESSORS

One way to minimize the amount of hardware needed to perform a particular application is to design the system and circuit blocks so that each block performs its part of the application in real time. Instead of ignoring the problem of I/O bandwidth, each chip is designed to eliminate the problem by accepting a standard video stream as input and generating a standard video stream as output. In this way, the I/O requirements do not increase as the amount of processing increases. The blocks can simply be inserted in the video stream and generate a modified video stream that can be utilized by the next processor (figure 9). As more processing functions are required, more blocks are added to the system. One obvious benefit of all this is that the frame buffers are not needed at all (for all of the other circuits discussed an external frame buffer or storage device is needed).

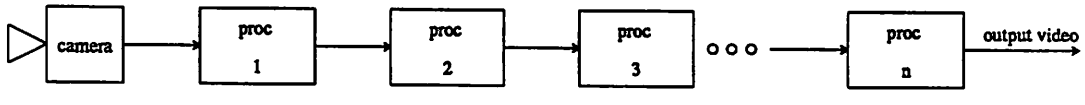



**Figure 9. Possible Connection of Real-Time Processors**

The net result of having all chips operate in real time is that each processing element must perform all data buffering for the algorithm that it is executing. It may appear that all that has been done is to

move the storage problem from off chip to on chip. This may be somewhat true, but each algorithm may only need to buffer part of the video image instead of the entire image. In addition, the way in which data is accessed from the buffer can be exploited to simplify the buffer design. For example, to perform 2-dimensional convolution requires the storage of at least $n-1$ lines for an $nxn$ convolution. If the architecture is chosen properly, the data required by processing elements are exactly one video line apart. For this architecture (different from Kung's), the storage requirements can be met with $n-1$ line delays. It should be noted that as $n$ increases, only the number of line delays increases, not the rate at which data must be read from the line delays. Although the bandwidth of each line delay does not increase, the bandwidth of the entire data buffer increases because the number of line delays increased.

This approach works very well for local operations that require limited storage capacity. For more global operations (e.g. contour tracing), the total amount of required storage is larger. The contour tracer has to store an entire frame on chip, but the image is only a single bit wide and thus the storage requirements are not as severe and a 128x128 buffer could be put on a 4μ NMOS chip. With a more advanced technology, the amount of storage that could be put on one chip could be increased.

Each video processor block is designed with a dedicated architecture that contains only the circuits that are needed for its particular function or class of functions so that no excess hardware is included. In essence this means that as many processing elements as are needed to perform the algorithm in real time are included on the chip, but no more. For example, to perform the histogram operation, which requires only a RAM and an incrementer, a chip with only a RAM and incrementer would be designed. To ensure that the hardware is fully utilized, the inter-processor communication is customized to prevent bottlenecks.

## 4.5 REFERENCES

[1] Reeves A. P., "Parallel Algorithms for Real-Time Image Processing", *Multicomputers and Image Processing*, Academic Press, New York, 1982, pp 7-18.

[2] Preston K. Jr, "Cellular Logic Algorithms for Gray-Level Image Processing", *Multicomputers* and *Image Processing*, Academic Press, New York, 1982, pp 135-148.

[3] Nukiyama T., "A VLSI Image Pipeline Processor", 1984 *ISSCC Digest of Technical Papers*.

[4] Shaw, D. E., "The NON-VON Supercomputer", *Technical Report*, Department of Computer Science, Columbia University, 1982.

[5] Mori T., "A Micro-Programmable Realtime Image Processor", 1986 *ISSCC Digest of Technical Papers*.

[6] Kung H. T., Song S. W., "A Systolic 2-D Convolution Chip", *Multicomputers* and *Image Processing*, Academic Press, New York, 1982, pp 373-384.

[7] Kung H. T., Picard R. L., "One-Dimensional Systolic Arrays for Multidimensional Convolution and Resampling", *Vlsi* for *Pattern Recognition* and *Image Processing*, Springer-Verlag, Berlin, 1984.

[8] Kanuma A., et. al., "A 20MHz 32b CMOS Image Processor", 1986 *ISSCC Digest of Technical Papers*, pp. 102-103.

[9] Kung, S. Y., "On Supercomputing with Systolic/Wavefront Array Processors," *Proc. of IEEE*, Vol. 72, No. 7, July 1984.

[10] Potter, J. L., editor, *The Massively Parallel Processor*, The MIT Press, Cambridge Mass., 1985.

# CHAPTER V DESIGN TECHNIQUES FOR FAST CIRCUIT DEVELOPMENT

## 5.1 INTRODUCTION

It has been pointed out that custom processors can achieve a higher throughput in a smaller area than general purpose processors by using architectures dedicated to particular applications. To avoid prohibitive design times for these custom circuits, two primary techniques have been employed. First, the hardware was designed and utilized in a hierarchical manner. Secondly, CAD tools were developed and used to assemble hardware at one level in the hierarchy from lower level hardware.

The techniques described made it possible to develop a set of eight custom chips in only 1.5 man years.

## 5.2 THE HARDWARE HIERARCHY

The hardware hierarchy is four levels deep with hardware at one level of the hierarchy being composed of hardware at lower levels. At the top level of the hierarchy are the image processing systems such as image recognition, image enhancement, etc. At the next level are chips that perform basic image processing algorithms such as convolution and contour tracing. The third level contains macrocells, or large circuit blocks. The blocks include RAMs, ROMs, arithmetic data paths and controllers. The last level contains the basic cells, such as: registers, adder cells and ROM cells.

The hardware at each level was designed to be re-used as often as possible in the hardware at the next higher level. The chips are complete functional blocks such as linear and logical convolvers, look-up tables, sorting filters and contour tracers that can be used in different image processing systems or to perform multiple functions within a single system. For example, the 3x3 sorting filter can perform both

median filtering for noise rejection and maximum-difference high-pass filtering for edge extraction. The same macrocells are used in many different chips. There are three major types of macrocells: storage elements (ROMs, RAMs and line delays), bit-sliced data paths and controllers. Every chip is composed primarily of these three macrocells. Finally, the same basic cells are used in different macrocells. For example, the same line delay was used in 3 different chips and the basic arithmetic blocks were used in the data paths of 5 different chips.

## 5.3 CAD ISSUES

In addition to re-using hardware as often as possible, CAD tools were used to assemble some of the hardware at one level in the hierarchy from hardware at a lower level. A data-path generator automatically assembles bit-sliced data paths from basic cells and a module generator is used to create tiled macrocells from the basic cells. At a higher level, a place and route tool is being developed to assemble the chips from macrocells.

The worst problems encountered in the layout of the chips are the assembly of blocks with many details that must be handled individually. These include the assembly of ROMs and PLAs that may have thousands of cells that must be programmed. Also in this class are data paths with complex signal connections. Random circuit routing with many nets is also a job that is often performed inefficiently and with many errors. The assembly of these circuits is very tedious, time consuming and error prone. Further, small modifications can mean that the entire circuit block must re-assembled so that the previous assembly time is wasted.

There are also some circuit assembly problems which are quite easily handled. Regularly tiled macro-cells and simple data paths can be created quickly and modified easily by hand. For example, a data path made out of a stack of bit slices (with no additional signal routing) can be created with a single Kic command. Regular structures such as a RAM array can similarly be created with a single Kic command. Small data busses can often be routed quickly by hand.

There is also a vast difference in the complexity of the software tools required to perform these tasks. The module generation problem is the simplest because the entire circuit assembly is solely a func-

tion of the cell design. With the basic cell parameters and a description of the way in which the cells are tiled together, the circuit can be assembled. No optimization or complex algorithms are needed. The data path generation is of slightly greater complexity because the program must place and route the cells. However, a simple data path compiler can be developed quickly if complex optimization is not performed. The program should perform the most tedious and error prone parts of the task and let the user make the higher level decisions. Finally, the most difficult task is the general placement and routing for the chip assembly. This program requires a fairly complex human interface and data base system. In addition, the more general nature of the problem makes the development time correspondingly longer as more general (and robust) algorithms must be developed.

Although it would be desirable to have the entire chip design automated, some tools must be implemented first due to relatively high development costs for some software tools. In light of this, it was decided to first implement the tools which provide the greatest utility for the lowest cost. Therefore, a ROM generator was developed first; in actuality it was just adapted from a filter bank generator [1,2]. Modgen [3] , a more general module generator, was developed later to allow the assembly of any tiled macrocell. The data-path generator was developed when it became apparent that data paths with more complex signal routing were required and that these new data paths would require too much human effort to assemble by hand. Finally, the placer and router (FLINT [4]) is currently being developed and is nearly finished.

Although much of each chip was hand assembled, automating the most time consuming and error prone aspects of the design reduced the total development time significantly. Even the assembly of a fairly complex chip from macro cells could be completed in a few days.

## 5.4 MODULE GENERATION

Because ROMs appear in many circuits and are exceedingly difficult to correctly program by hand, it became apparent that a ROM generator would be extremely valuable. The program used was simply taken from the filter bank generator and modified for slightly different input and output registers and row buffers. The program explicitly specifies the tiling process as a function of the circuit parameters. The circuit parameters include the number of words, number of column decoders, number of outputs and the

binary contents. In addition, programs were written to generate the binary ROM contents for each particular chip because this was also a tedious task. For the linear convolver, a program determines the micro code from the desired impulse responses. A program was used to convert an arbitrary point-wise function into binary ROM contents for look-up-table ROMs. For development of the contour tracer, a program generates the FSM contents from a state transition table. Modgen was used to generate the PLAs and the FSM that were used in the feature extractor.

## 5.5 DATA-PATH GENERATION

### 5.5.1 Introduction

The data-path generator was developed to assemble bit-sliced data paths made up of cells according to the user's specified organization. The user specifies the organization in terms of some number of bit slices (of the same or different types) and the way in which the slices are made of some connection of basic functional blocks such as adders, registers, etc. All routing between cells is handled automatically. Data regarding the terminal locations, terminal layers and the cells bounding box size are determined by the "parse" program and stored in the file, "celldata", that is read by the generator.

Because the data-path generator was developed for the image processing chips it will be discussed in some detail. The data-path generator was used to generate the data-paths in the sorting filter, the non-linear low-pass filter, the feature extractor, a histogram chip [5], and the small storage array in the programmable convolver controller.

### 5.5.2 Overall Organization

The data path is made up of some number of slices. These slices stack from top to bottom (figure 1) and data flows from left to right (or vice versa). There are two general classes of slices, those that contain data signals and those that do not. Slices without data signals are typically top or bottom slices that connect GND, Vdd, and the clocks to the bit slices. Data slices contain blocks which are routed according to the user-specified description. In addition, the user can specify the different slices which make up a data path. It is possible for the user to use as many slice types as are needed. For example, one might have a top slice (CNT for control), an MSB, EVEN and ODD (for optimized ripple carry adders) and a

bottom slice (GND).

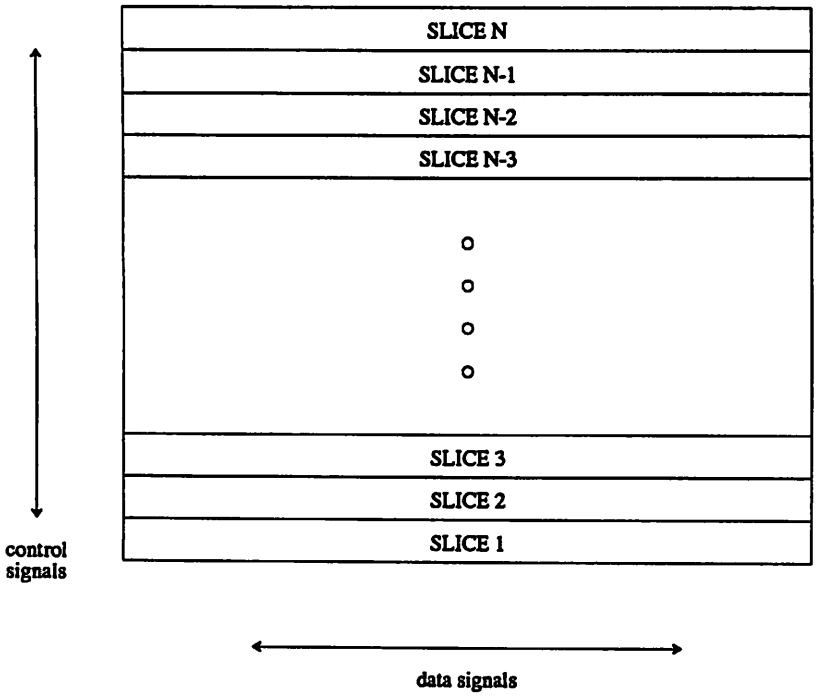



**Figure 1. Data Path**

Each slice is made up of blocks (figure 2). Typically these are basic building blocks, such as, adders, shifters, registers, etc. Again, there is no restriction on the number of blocks. The actual implementations of a block for each slice are called cells. The user simply needs them to be designed and declared in the "celldesc" file. This file describes the Kic cells which are to be used in each slice of a block.

All cells in a given slice must have the same height to make the global routing channel rectangular. The width of cells for different blocks in the same slice can be different, but the width of all cells of a particular block must be the same.

All cells used in the data slices must have signal terminals on the left or right side of a cell in either polysilicon (POLY) or diffusion (DIFF). Control terminals must be on the top or bottom of the cell in metal. The slices without data signals may have two types of terminals. The first are those on the left or right of the cell which should be connected to the terminals of the next cell on the right or left, respectively. These terminals are simply extended to the next cell. The second type of terminal is found on the

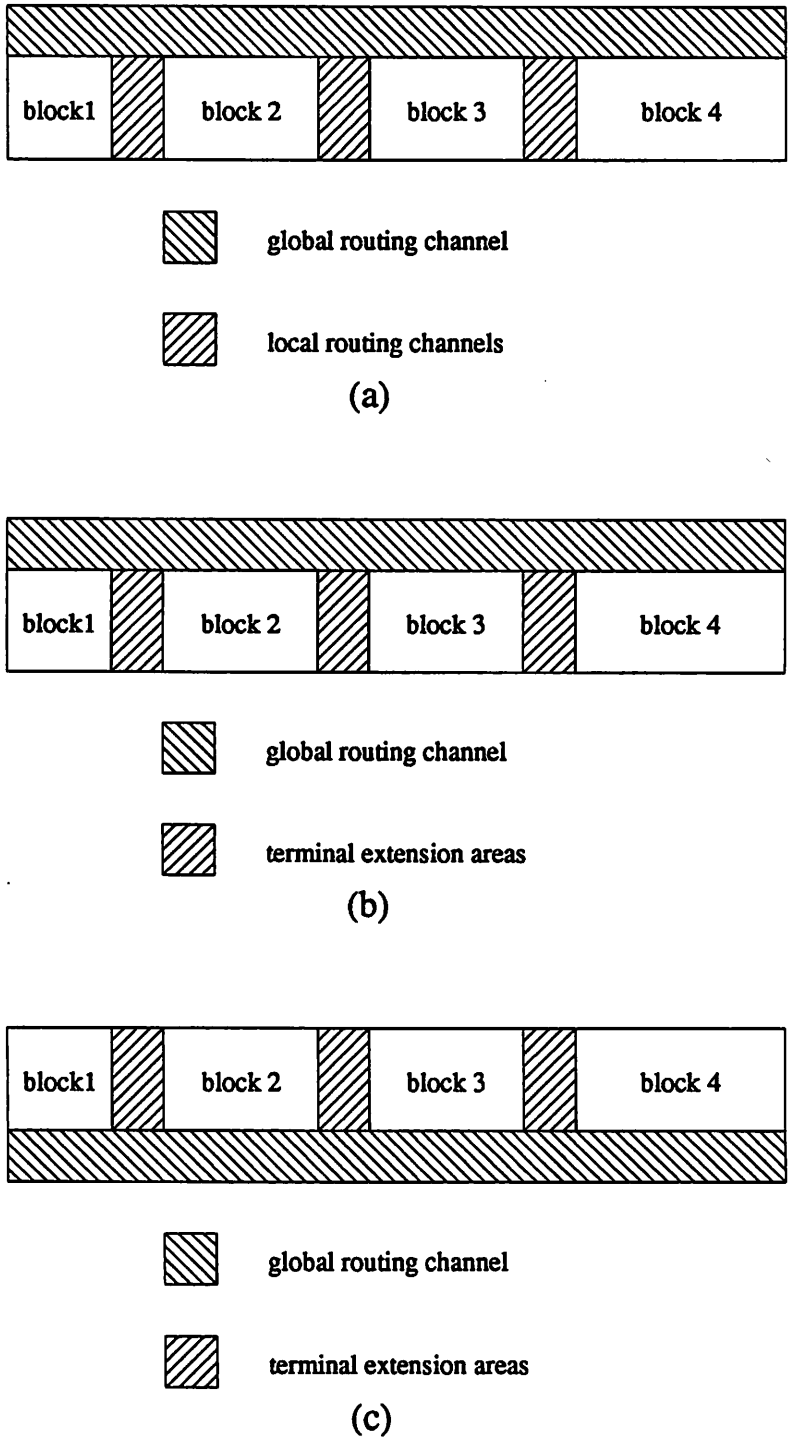



**Figure 2. Data Slice (a), Top Slice (b), Bottom Slice (c)**

top or bottom of the data path (the top of the top most slice or the bottom of the bottom most slice) in poly. These terminals correspond to control inputs and outputs and are routed according to the user specified description.

The user describes the placement of the blocks and the way in which the terminals of the blocks should connect. Because the data path is bit sliced, this information need only be specified once for all slices containing data signals. The user may also specify how any signals which enter/leave the top or bottom of the data path should be connected. These signals are typically control signals (e.g. the adder carry input). For example, the user may want to connect the adder carry in to an invert signal to obtain a true twos-compliment subtraction. All connections are made within a single slice. That is, it is not possible to have terminals of one slice connect to terminals of another slice. If inter-slice connections are desired (such as in a shifter), a cell has to be designed to make the connections.

### 5.5.3 Data-path Description

The user describes the data path in an input file (figures 3 and 4 give the syntax diagrams). There are three main parts to this description. The first part contains the lists of external nets (nets which leave or enter the data path for connection to another circuit block). A list for each side gives the net name, a net number and a cable name. The net number and the cable name are used by FLINT and are passed to the HDL file. In addition to external signal nets, the user declares external control nets which should be handled differently than signals. These include Vdd, GND and the clocks. The program checks for the any occurrences of "control" terminals on the boundary of the data path and marks the position of the terminals appropriately for FLINT in the HDL file.

The next part of the data-path description is a list of the slice types which gives the name of the slice type and whether that slice has data signals or not. Finally, comes the description of how the slices are constructed.

### 5.5.4 Slice description

The slice description was chosen to make things simple for the user and the data-path generator. Basically, this part of the description declares a list of blocks and the connections of any routed terminals on each block. The basic format is "block (terminal=net, terminal=net, ...)". Since most blocks have a single input and output, the connection of these signals can be described easily through use of a redirect symbol, ">". This symbol is used similarly to the UNIX pipe and redirect. The symbol ">" used before a

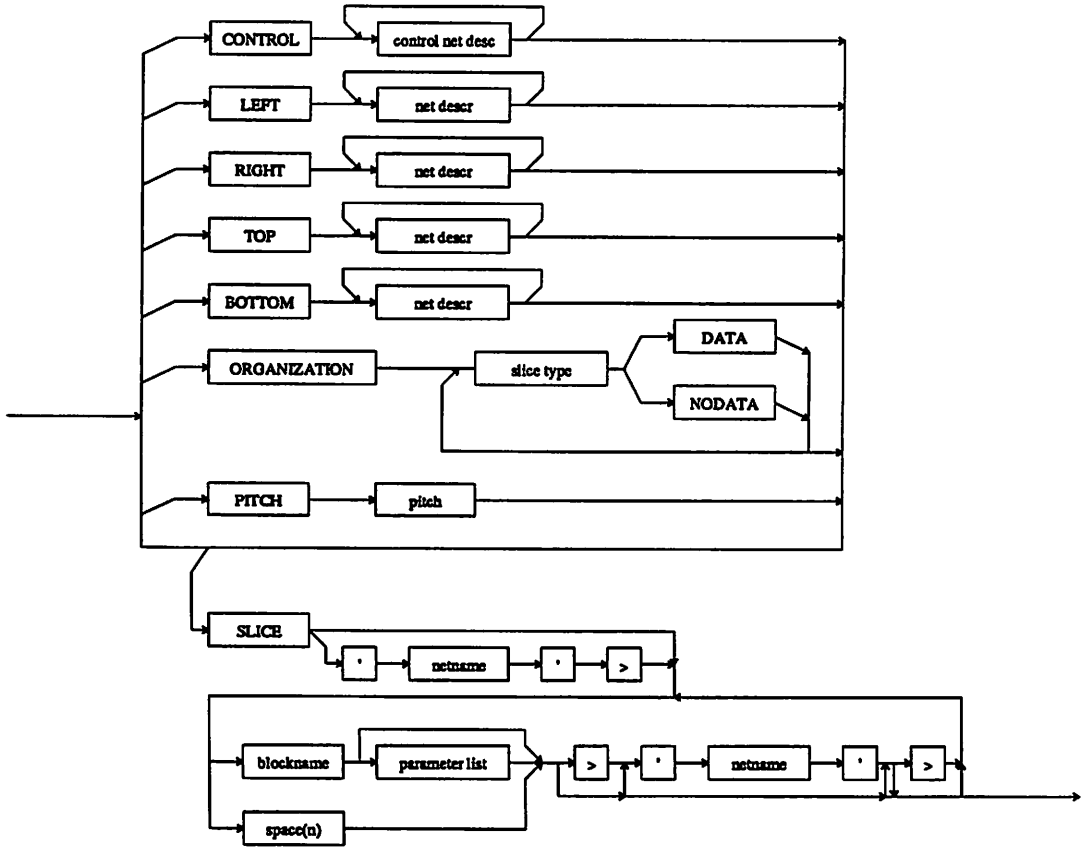



**Figure 3. Syntax Diagram for the Data Path**

block declaration indicates where the standard input of a block (the terminal labeled "in") should come from and where the standard output (the terminal labeled "out") should go when used after a block declaration. The standard input can be connected to a net or to the standard output of another block. Similarly, the standard output can be connected to a net or to the standard input of another block. If a block has more than one input or output, these connections must be described explicitly in a parameter list. This setup is completely general, in that, any connection of blocks with any number of terminals can always be made. The defaults are simply included to prevent having to explicitly specify every connection.

Connections to terminals on the top and bottom of the data path are specified by putting "T/terminal=net" or "B/terminal=net" in the parameter list, respectively.

The slice description not only gives the information of how blocks are connected but also of the

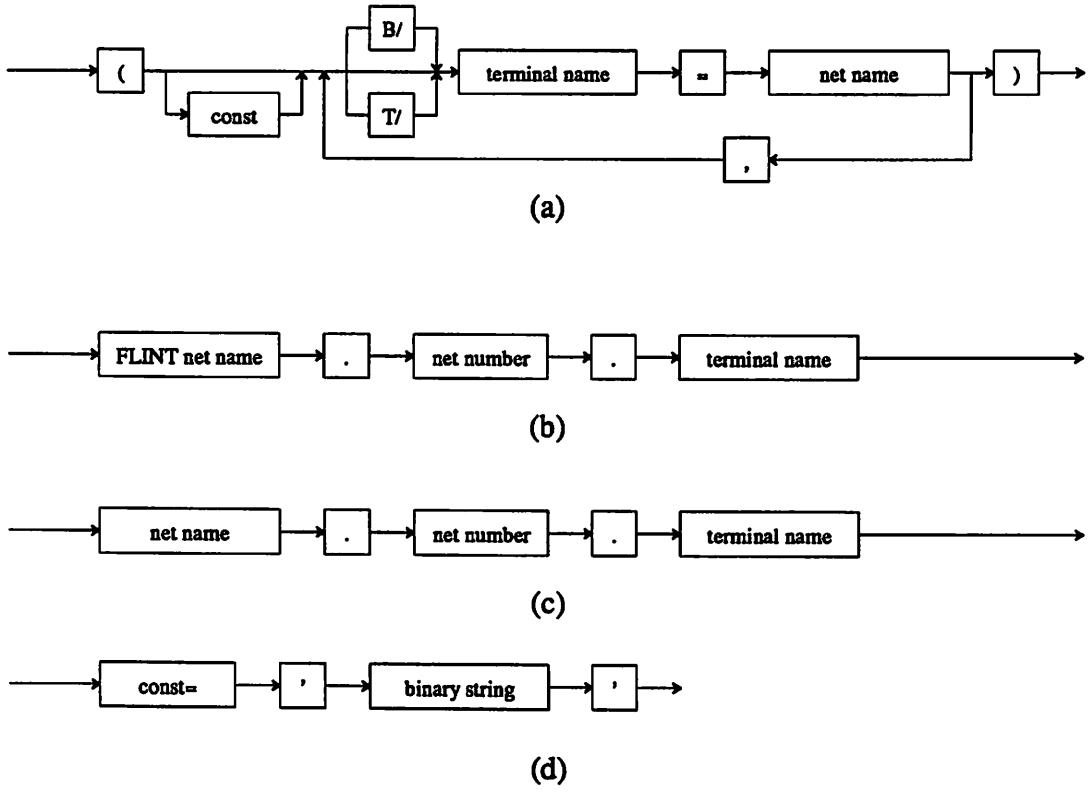



**Figure 4. Syntax Diagrams for "parameter list" (a), "control net desc" (b) "net desc" (c), and "const" (d)**

physical placement of the blocks. The blocks are placed in the same order they are described in in the input description. It would be possible to have the program choose a block placement which minimizes the number of tracks needed for the channel routing. This creates a problem if the user has a different optimization criteria than the program. For example, the user may not care if the channel route required 5 or 7 tracks but may care a great deal if two critical blocks are put on opposite ends of the data path.

### 5.5.5 Creating Extra Space

A function is available to force more space between blocks. It can be used to create space in the circuit for something that the data-path generator can not handle such as inter-slice routing. If "space(n)" is inserted between two blocks, n Kic units will be added to the space between the two blocks. 100 Kic units is one lambda. The user also has some control over the pitch of the bit slices. If the minimum slice height is less than the user declared pitch, the slices are simply spaced out to achieve the desired pitch. This option makes it possible to attach a data path to another circuit which has a different bit pitch.

### 5.5.6 Constants

The previous discussion has assumed that all data bit slices have identical blocks and connections. If this were the case, it would be possible, but not very easy to implement constants in the data path. A constant parameter can be specified in the parameter list which tells the compiler how to modify the block in each slice (not each slice type). For example, suppose for the block counter, two counter cells exist, count.0 (loads 0) and count.1 (loads 1). To generate a counter which loads the value "10011", one would put "counter (const=10011, ..." in the SLICE description.

### 5.5.7 Capabilities

To improve the final circuit, some degrees of freedom are given to the cell designer. More than one terminal can exist for a given signal. Typically, a terminal in each layer (POLY and DIFF) could be brought out. With this extra freedom, the generator can more likely abut two blocks or choose routing layers to avoid metal or buried contacts to minimize channel width. Also terminals can be defined which are larger than the minimum conductor width, so the generator can choose the point to contact the terminal.

Handling these extra capabilities is not too difficult when there are few nets to be routed, but makes general routing more difficult. For example, to check if a river route is possible is fairly easy when each net has only two terminals and only one layer. Adding the extra terminals and layers (and the possibility of making buried contacts) makes this problem non-trivial.

### 5.5.8 Routing Strategy

There are basically five routing problems which must be solved. The nets at the top and bottom of the data path must be routed. In these cases, all terminals are on one side of the routing channel. There are three distinct routing problems in the data slices. First, there is the routing of global nets above the blocks. Since these nets must cross metal control lines, the nets are routed in poly. Second, there are the nets that must connect from a cell to the global routing channel. Finally, there are the nets that are local, that go from one cell to another without entering the global routing channel. To simplify the problem and produce a small routing channel, certain decisions were made. The nets that enter the global routing

channel are routed in metal in a river fashion and local nets are routed in a river in poly or diffusion. Although the local routing could be easily solved by using a channel router, it would often produce poor results for typical channels with only one output and one or two inputs.

#### 5.5.8.1 Global Routing

The global routing problem is simplified since all terminals are on one side of the channel. This means that there are no vertical constraints and the channel can always be routed in the minimum number of channels using a very simple algorithm. To minimize the routing area, the tracks are not evenly spaced because the contact pitch is greater than the poly pitch. Alternating tracks have their contacts on opposite sides of the poly.

#### 5.5.8.2 Local to Global Routing

Routing of the signals to the global routing channel is a river route problem with one terminal free to move. These are routed in metal since the global routing is done in poly and these nets must cross the global nets to make connections. Because all nets leave the local routing channel on the same side of the channel, the channel width is determined by the density or simply the number of nets.

#### 5.5.8.3 Local Routing

The local routing is a fairly complex problem because there are obstacles (metal contacts to connect terminals to the global routing channel) and the terminals can be in poly or diffusion. Further, complications arise with the possibility of connecting a poly terminal to a terminal in diffusion because a buried contact is needed. The program checks for obstacles and proper design rule spacings to minimize the routing channel width. To simplify the routing algorithm, only river routing is allowed. This is not really a major restriction since most local routing channels have at most a single net.

### 5.5.9 Datapath Generator Operation

The program first reads the user input file (the data path description) and the cell data file (lists of cells, bounding box sizes, terminal locations, etc) and creates the linked list data structures. The placement and routing is done for each slice type independently in a two pass fashion.

On the first pass, the area required for all the local routing channels is determined, but no geometries are actually created. This allows for differences in the area required for the local and global routing between the different slice types. On the second pass, the block placement is done, separating blocks by the maximum area required (over the slice types) for the local routing. After each block has been placed, all routing in the local channels is performed and the geometries are created. When all blocks for the slice have been placed, the global routing above the blocks is performed, since now the position of all global terminals is known. All terminals on the top of the cells are then extended to cross the poly global routing channel. This procedure is followed until all slice types have been created. Then the entire data path is created out of these different slice types and the signals in each slice are labeled for simulation. At this point, blocks modified by the constant parameter are inserted.

### 5.5.10 Cell Parser

In order to relieve the cell designer and users of the system from characterizing the cells in a description file, a program was written to create the cell description file automatically. The program "parse" reads the file "celldesc" to see which cells have been defined. It then parses each Kic file to determine its bounding box and the position and layer of all terminals of the cell. The data is written to the file "celldata" for use by the "compile" program. This process need only occur when a new cells are added to the library, not every time a data path is generated.

The bounding-box information is computed as the boundary of all geometries in the cell. The terminals are determined from information in the "NS" layer that is put in by the cell designer. A box in the "NS" layer with a side that touches the side of the cell indicates the terminal boundaries. A label in the same layer that is inside the box indicates the name and mask layer of the terminal. This information also creates documentation in the cell.

The input file "celldesc" is a list of all cells used for each block. The format is:

```
BLOCK name
        SLICE type cellname
        SLICE type cellname
              ...
        SLICE type cellname

BLOCK name

        ...
```

The types are defined by the user and only have meaning in the organization description in the data path input file. The exception to this is the type "CELL". This is the default type and the cell defined as "CELL" will be used if the type in the organization description does not exist for that block. For example, the delay block uses the same cell for EVEN, ODD and MSB slices. So, instead of defining all three types to have the same cell, only the "CELL" type is defined.

The operation of the parser is fairly simple. It opens each file and finds all boxes. It finds the bounding box corners from the minimum and maximum of the box boundaries in each dimension. It then scans the symbolic layer (NS) for terminal names and locations. To help the data-path generator, the terminal information is stored by name, side of the bounding box the terminal is on, the layer it is in and the boundaries (either in X or Y) of the terminal.

The parser assumes that "nice" files will be read; that is, in the format written by Kic. Kic writes files with all geometries for a given layer grouped together and with all layers in the order specified in the .KIC file. Since all cells used in this system were developed with Kic, this is a reasonable assumption. If files that do not meet these requirements are to be used, a program can be written to make them "nice" or they could simply be read and written by Kic.

The currently used "celldesc" file is given below. It shows the blocks that are defined.

### 5.5.11 Cell Description File

```
BLOCK shiftright
      SLICE CELL shiftright
      SLICE GND  shiftright.gnd
      SLICE CNT  shiftright.cnt
      SLICE MSB  shiftright.msb

BLOCK const     /for introducing hardwired constants/
      SLICE CELL const
      SLICE GND  const.gnd
      SLICE CNT  const.cnt

BLOCK sideright /interface for FLINT/
      SLICE CELL sideright
      SLICE GND sideright.gnd
      SLICE CNT sideright.cnt

BLOCK sideleft /interface for FLINT/
      SLICE CELL sideleft
      SLICE GND side.gnd
      SLICE CNT side.cnt

BLOCK maxselect          /mux for max circuit/
      SLICE CELL maxsel
      SLICE GND maxsel.gnd
      SLICE CNT maxsel.cnt

BLOCK invert-clear  / inverter with clear/
      SLICE CELL invert-clear
      SLICE GND  zero-one.gnd
      SLICE CNT  zero-one.cnt

BLOCK zero-one  / cell to generate either low or high output/
      SLICE CELL zero-one
      SLICE GND  zero-one.gnd
      SLICE CNT  zero-one.cnt

BLOCK smlatch   /small static latch without TS buffer or input inverter/
      SLICE GND staticlat1.gnd
      SLICE CNT staticlat1.cnt
      SLICE CELL staticlat1

BLOCK latch     /static latch with TS buffer for convolver/
      SLICE GND staticlat.gnd
      SLICE CNT staticlat.cnt
      SLICE CELL staticlat

BLOCK zero
      SLICE CELL zero
      SLICE GND zero.gnd
      SLICE CNT zero.cnt

BLOCK sort2     /produces max(a,b) and min (a,b)/
      SLICE MSB sort.o
      SLICE EVEN sort.e
```

```
        SLICE ODD sort.o
        SLICE CNT sort.cnt
        SLICE GND sort.gnd

BLOCK delay2    /2 input, 2 output register/
        SLICE GND delay2.gnd
        SLICE CNT delay2.cnt
        SLICE CELL delay2

BLOCK increment         /incrementer carry chain and sum/
        SLICE EVEN carrye
        SLICE ODD  carryo
        SLICE MSB  carryo
        SLICE GND  carry.gnd
        SLICE CNT  carry.cnt

BLOCK buffer2   /dual buffer for sort circuit/
        SLICE CELL buffer2
        SLICE GND buffer2.gnd
        SLICE CNT buffer2.cnt

BLOCK mux2to2 / switch circuit /
        SLICE GND mux2to2.gnd
        SLICE CNT mux2to2.cnt
        SLICE CELL mux2to2

BLOCK compare /comparator -- smaller than adder and inverter/
        SLICE CNT compare.cnt
        SLICE GND compare.gnd
        SLICE MSB compare.o.new
        SLICE EVEN compare.e.new
        SLICE ODD  compare.o.new

BLOCK TSbuffer          / non inv tristate buffer /
        SLICE GND busdriv.gnd
        SLICE CNT busdriv.cnt
        SLICE CELL busdriv

BLOCK buffer    / non inv super buffer /
        SLICE CELL buffer
        SLICE GND buffer.gnd
        SLICE CNT buffer.cnt

BLOCK mux2to1 / 2 input mux - no regenerating logic /
        / has 2 inputs -- 1 from control slice (delay)
                -- 1 for user override/
        SLICE CELL mux2to1
        SLICE GND mux2to1.gnd
        SLICE CNT mux2to1.cnt

BLOCK possataccum    /full register with pos saturation logic and feedback/
        SLICE MSB sataccum1
        SLICE CELL sataccum1
        SLICE CNT sat.cnt
```

```
    SLICE GND sat.gnd

BLOCK sataccum        /full register with saturation logic and feedback/
    SLICE MSB sataccum1.msb
    SLICE CELL sataccum1
    SLICE CNT sat.cnt
    SLICE GND sat.gnd

BLOCK satregister     /full register with saturation logic and feedback/
    SLICE MSB sataccum.msb
    SLICE CELL sataccum
    SLICE CNT sat.cnt
    SLICE GND sat.gnd

BLOCK possatregister  /full register with pos saturation logic and feedback/
    SLICE MSB sataccum
    SLICE CELL sataccum
    SLICE CNT sat.cnt
    SLICE GND sat.gnd

BLOCK posadder        /2's compliment adder - carry in from preceding block/
    SLICE EVEN naddce
    SLICE ODD  naddco
    SLICE MSB  naddco
    SLICE CNT  posaddersat.cnt  /contains saturation logic /
    SLICE GND  adder.gnd2       /has cin logic for 2's comp subtr/

BLOCK adder           /2's compliment adder - carry in from preceding block/
    SLICE EVEN naddce
    SLICE ODD  naddco
    SLICE MSB  naddco.msb       /different to handle saturation /
    SLICE CNT  addersat.cnt     /contains saturation logic /
    SLICE GND  adder.gnd2       /has cin logic for 2's comp subtr/

BLOCK delay1  / full register ph2-ph1/
    SLICE CELL delay
    SLICE CNT delay.cnt1  /passes msb to next block/
    SLICE GND delay1.gnd2       /flattened version of delay.gnd1/

BLOCK buff  / non-inverting buffer/
    SLICE CELL buff
    SLICE CNT buff.cnt
    SLICE GND buff.gnd

BLOCK delay-clear  / full register ph1-ph2 with clear/
    SLICE CELL delayc
    SLICE CNT delayc.cnt1 /passes msb to next block/
    SLICE GND delay1c.gnd1      /flattened version of delay.gnd1/

BLOCK delay  / full register ph1-ph2/
    SLICE CELL delay
    SLICE CNT delay.cnt1  /passes msb to next block/
    SLICE GND delay1.gnd1       /flattened version of delay.gnd1/
```

```
BLOCK minus1 /multiply by minus 1 -- invert /
        SLICE CELL invert
        SLICE GND invert.gnd
        SLICE CNT invert.cnt

BLOCK absvalue
        /control inputs -- zero and absvalue/
        SLICE MSB abszero.msb
        SLICE CELL abszero
        SLICE CNT abszero.cnt
        SLICE GND abszero.gnd1

BLOCK gain      /variable gain (1,2,4,8)/
        /control inputs shift1,shift2,shift3/
        SLICE CELL gain1        /flattened version of 'gain'/
        SLICE CNT shleft1.cnt  /flattened version of shleft.cnt'/
        SLICE GND gain3.gnd  /saturation logic/

BLOCK modify   / handles output format conversion for video/
        /control inputs negclip,binclip/
        SLICE CELL modify1   /falttened version of modify/
        SLICE GND modify.gnd
        SLICE CNT modify.cnt
        SLICE MSB modify1.msb      /flattened version of modify.msb/
```

### 5.5.12 Nonlinear Low Pass Filter Processor

This data path (see section 7.3.6.1 for a description and block diagram) performs the conversion from linear to non-linear low-pass filter. Overall it is very similar to the Sobel processor except that it has more complicated signal connections. Basically, this processor takes an all-pass and high-pass signal and generates the corresponding low-pass signal. If the magnitude of the high-pass signal exceeds a threshold, the low-pass signal is output. Otherwise the all-pass signal is output. Pads were added to the data path to make a complete chip for fabrication. The input description for this block and the plots follow in figures 5 and 6. Figure 5 shows the actual work performed be the generator. The generator placed the blocks shown in the figure and created all of the geometries shown. The full circuit plot is shown in figure 6.

#### 5.5.12.1 Nonlinear Low Pass Filter Input Description

```
/*
  non linear low pass filter processor
  5/2/85
*/

CONTROL
        Vdd.0.Vdd
        GND.0.GND
        ph1.0.phi1
        ph2.0.phi2

TOP
        nofilter.40.cable5
        binary.41.cable0

BOTTOM
        negclip.42.cable
        absvalue.43.cable
        invert.44.cable
        lpf.45.cable5

LEFT
        laplac_in.0.cable1
        allpass_in.10.cable2  /* input signals */

RIGHT
        out.20.cable3
        thresh.30.cable4   /* output signal and threshold */

ORGANIZATION
        CNT     NODATA
        MSB     DATA
        EVEN    DATA
        ODD     DATA
        EVEN    DATA
        ODD     DATA
        EVEN    DATA
        ODD     DATA
        EVEN    DATA
        ODD     DATA
        EVEN    DATA
        GND     NODATA

SLICE
        'allpass_in' > delay > 'allpass'

        /* generate low pass signal */

        'laplac_in'  > minus1 (T/'high'='cin1')
        > delay (T/'in'='cin1',T/'out'='cin2')
        > adder ('inb'='allpass',T/'cin'='cin2')
        > satregister > 'lowpass'
        'allpass' > delay > 'allpass1'
```

```
/* generate decision */

'laplac_in'
> absvalue (T/'carry'='cin3',B/absvalue=absvalue,T/zero=nofilter,B/invert=invert)
> delay (T/'in'='cin3',T/'out'='cin4')
> adder ('inb'='thresh',T/'cin'='cin4')
> delay  (B/'msb'='decision')

/* generate output */

'allpass1'
> mux2to1('inb'='lowpass',B/'in_to_out'='decision',B/inb_to_out=lpf)
>  delay
> modify (T/'binarycon'=binary,B/'negclip'=negclip)
> 'out'
```

**Figure 5. Non-Linear Low-Pass Filter Plot (high level)**

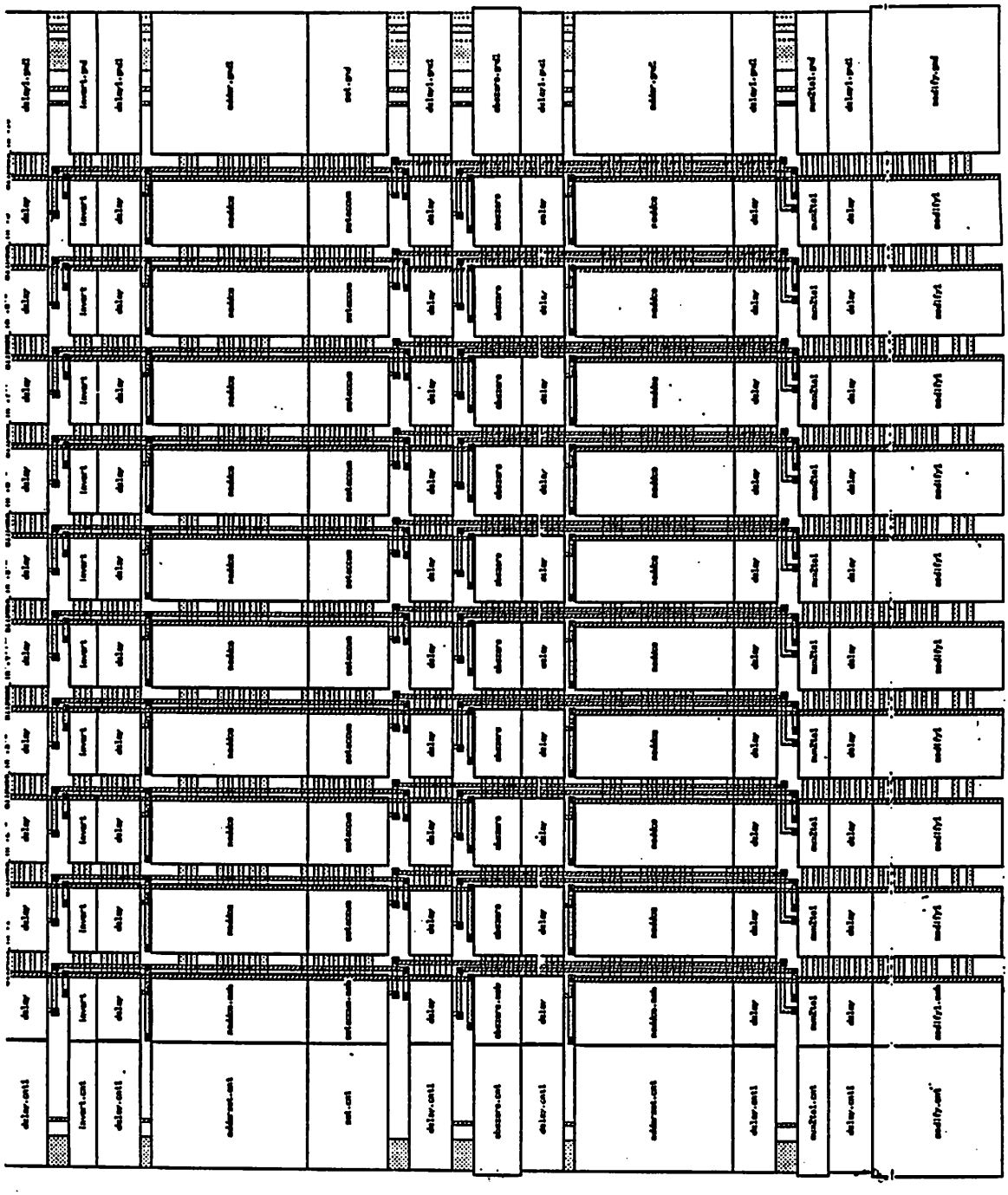
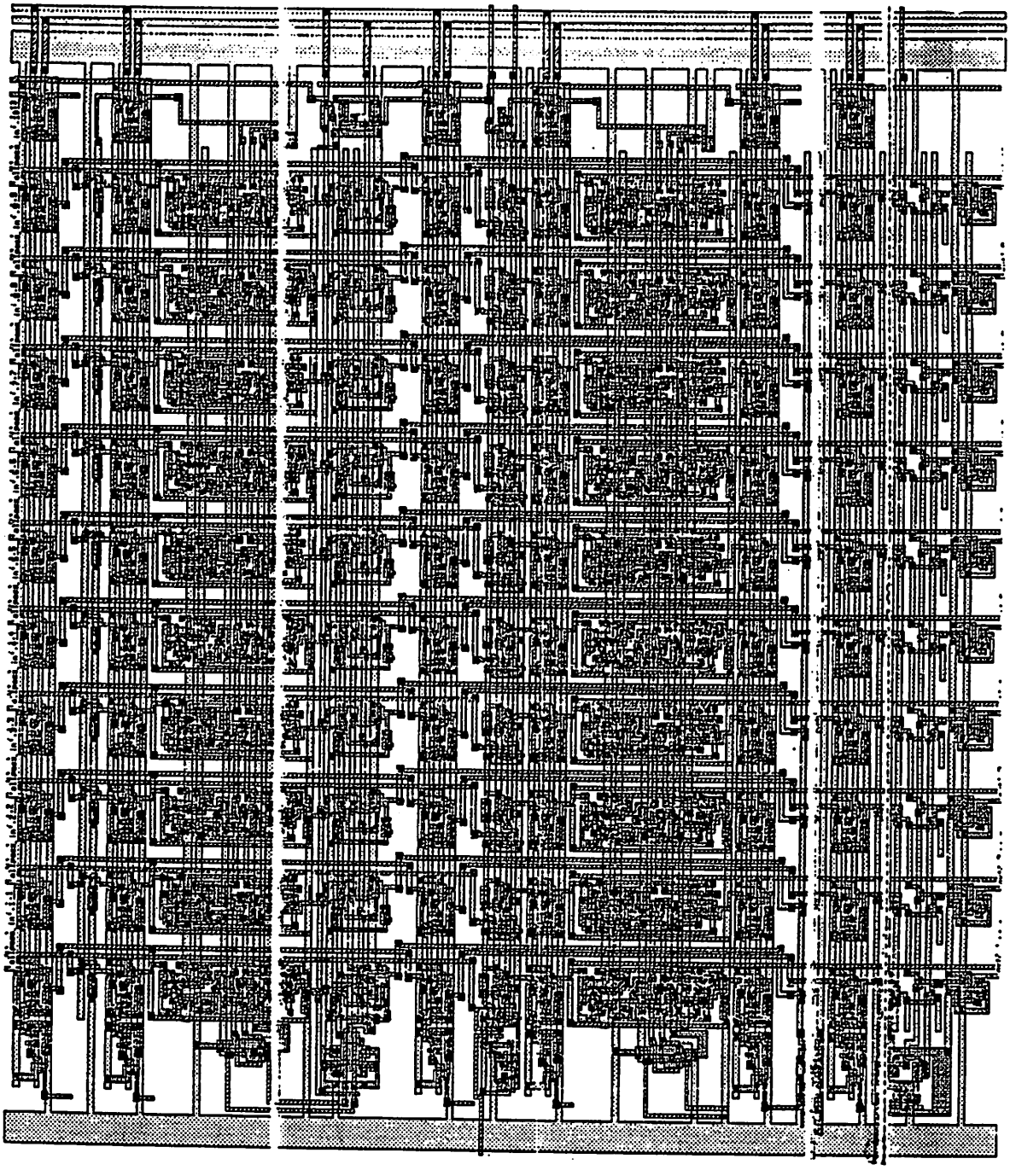


**Figure 6. Non-Linear Low-Pass Filter Plot (detailed)**



### 5.5.13 Program Documentation

The programs are broken into several major blocks. These are the cell parser, "parse", the user file parser, "compile", the placement and routing routines ,"route", "localroute" and distance" and the basic kic cell generation routines, "createkic". "datapath.h" contains the constant and structure definitions.

#### 5.5.13.1 Parse.c

The routine "nextcell" reads data from the cell description file until the next cell declaration is found. It then will try to open this file and return the pointer to it. It also copies the slice type to the output file.

"scansymboliclayer" finds all boxes and labels in the symbolic layer. It then matches the boxes and labels and writes the required information to the output file.

"layer" extracts the layer information from the label.

"findside" determines which side of the cell bounding box the terminal is on.

"readlabel" reads a label from the kic file.

"findboundbox" reads all boxes in the KIC file and computes the coordinates of the cell"s bounding box from that.

"readbox" reads a box from the kic file.

#### 5.5.13.2 Compile.c

This file contains routines to read in the input files and create the data structures so that the datapath can be generated.

##### 5.5.13.2.1 Parsing routines

"parseinput" scans the user input file for the eight main fields and calls routines to read in the data.

"readcelldata" reads the "celldata" file and generates the linked lists of blocks, cells and terminals.

"checkerrors" checks for some errors during the assembly process.

"netnamematch" compares the user terminal name with that in the kic cell.

"readinput" parses the slice description of the user file and creates a list of strings. Comments are ignored.

"readslice" parses the list of strings created by "readinput" and checks for syntax errors and creates the linked lists of user blocks, nets and global nets.

"readparameters" reads the explicit and implicit net connections specified for a user block and adds these to the database.

"fixnetname" gets rid of the optional single quotes from net names.

"findkeyword" determines if a string is a keyword or not.

"checknets" checks for nets with only one terminal and also determines if nets exist only on the right or left of the datapath so that they can be handled efficiently.

"findslicetypes" reads the ORGANIZATION description and generates a list of the slice types used.

#### 5.5.13.2.2 Data Storage and Retrieval Routines

"initstructs" initializes the linked lists.

"addnets" adds the nets listed in the LEFT, RIGHT, TOP, BOTTOM or CONTROL fields to the net list.

"findterm" returns the pointer to the terminal with a given name and layer in a particular slice and block.

"findterm1" returns the pointer to the terminal in a given cell ,layer and name.

"newterm" adds a terminal to the linked list.

"newnet" adds a net connection to the list for a user block.

"findnet" returns a pointer to a net for a user block.

"newbitslice" adds a slice type to the slice type linked list.

"findbitslice" returns the pointer to bit slice with a given name.

"findcell" returns the pointer to the cell of a block.

"findblock" return the pointer to a block.

"newlocterm" adds a net to the list for a user block that connects to a terminal on that block.

"findlocterm" returns the pointer to terminal on a block with a given name.

"newuserblock" adds a new user block to the list.

"newblock" adds a new block to the list.

"newcell" adds a new cell to the list.

"newconst" adds a new constant to the constant list.

"addblockterm" adds net connections for a block to the appropriate lists.

#### 5.5.13.3 Route.c

This program contains routines to do the routing and placement.

"assembledp" coordinates all these routines. In the first pass the sizes of all routing channels are determined. On the second pass, instances are created and the routing is performed.

"extendcontrol" extends all terminals on the top of a cell across the global routing channel.

"labelexternnets" labels the external nets in the datapath and generates HDL file entries.

"routeLRnets" puts in the geometries for nets that exist only on the left or right of the datapath.

"outputnet" stores net and terminal information for global routing.

"routeglobal" performs the global routing and creates the geometries.

"addexternnet" adds an external net to the linked list.

"labelnet" saves information regarding the signals in each slice so that they can later be labeled.

"wireextend" extends all wires on a side of a cell a certain distance. This is used for extending control over the global routing channel and extending control in the NODATA slice types.

"routelocal" computes the distance necessary between blocks for routing or puts in the geometries. This routine just calls other routing routines.

"routelocalmetal" routes signals from the blocks to the global routing channel in metal. Actually this routine just computes the size of channel needed and the list of nets.

"abutt" determines if two cells can be abutted or not.

"sort" sorts the list of metal nets for routing.

"routemetal" creates the geometries for the metal route.

"metalcont" creates a metal to POLY or DIFF contact.

"insertconstants" puts in cells that were modified by constants.

"slice_global_route" collects all data slice nets and routes them. The control wires are extended to the next slice.

"topbot_global_route" handles the routing of nets on the top and bottom of the data path.

"outputleftnets" adds nets on the left side of the data path the global routing list.

"genhdl" opens the HDL file and writes the header information.

"gentopbotnets" adds nets on the top or bottom of the data path the global net list.

"findcontrol" adds nets of type control to the external net list.

#### 5.5.13.4 localroute.c

This file has some of the local (POLY, DIFF) routing routines.

"routelocPD" river route local nets between blocks

"routelocal1" creates the geometries for the route performed in "routelocPD".

#### 5.5.13.5 distance.c

This file contains some routines that compute the minimum spacings between nets in the local routing area.

"distance" determines the channel size (space between blocks) needed for a local route.

"findy1y2" determines the boundaries in the Y dimension of a net.

"localchansize" returns the channel size needed to route 1 net and the position of the net in the channel. Checks for constraints.

## 5.6 REFERENCES

[1] Ruetz, P. A., et al., "Computer Generation of Digital Filter Banks," *ISSCC Digest of Tech. Papers*, Feb. 1984, pp. 20-21.

[2] Ruetz, P. A., et al., "Computer Generation of Digital Filter Banks," *Trans. of CAD* for *Integ. Circ.* and *Systems*, April 1986.

[3] Ruetz, P. A., et al., "Automatic Layout Generation of Real-Time Image Processing Circuits," *Proc. of ISCAS*, May 1986.

[4] Rabaey, J. M., et al., "An Integrated Automated Layout Generation System for DSP Circuits," *IEEE Trans. on CAD*, Vol. CAD-4, no. 3, July 1985.

[5] Richards, B., "Design of a Video Histogrammer Using Automated Layout Tools," *UCB M.S. Report*, to be published.

# CHAPTER VI CIRCUITS

## 6.1 INTRODUCTION

As noted in the preceding chapter the hardware at one level in the hierarchy is re-used to make hardware at higher levels. In this section the basic cells will be discussed and the extent to which they can be re-used in the macro cells examined. Each macro cell can re-use the basic cell hardware to a different extent. Many data paths can make use of a rather limited cell library while the storage circuits tend to require a fair amount of new circuit design for each new storage macro cell. The controllers lie somewhere between these two extremes. The degree to which basic cells can be re-used depends upon the constraints put upon the basic cells.

Generally, circuits that are not in the critical path can use standard size transistors. The use of standard device sizes makes it possible to quickly layout new circuits. Typically, pull-down devices with $\frac{W}{L}=\frac{8}{4}$ are used in conjunction with depletion pull-up devices with $\frac{W}{L}=\frac{4}{8}$. This basic inverter is very compact. Of course, if very low power dissipation is desired, the pull-up current should be minimized. For high speed circuits, the width of the two devices could be increased.

Mead and Conway [1] design rules supported by MOSIS are used throughout. The use of these standard rules made it possible for many of the basic cell designs to survive several changes in the minimum feature size without modification.

## 6.2 COMMONLY USED CIRCUITS

Rather than show all circuits at the transistor level, it is convenient to represent some commonly used circuits more abstractly. The common register (figure 1) and several buffers (figure 2) are used in many different circuits and will not be drawn at the transistor level. Typically, the output depletion pull-

up device has $\frac{W}{L}=\frac{4}{4}$ and the output enhancement pull-down device has $\frac{W}{L}=\frac{16}{4}$. For heavily loaded output nodes, the width of the two output devices is doubled.

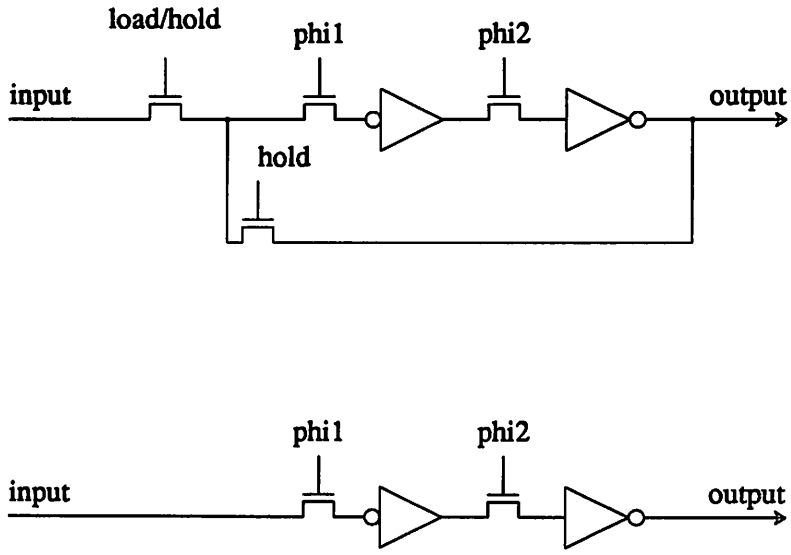



**Figure 1. Registers, load-hold register (top), load-only register (bottom)**

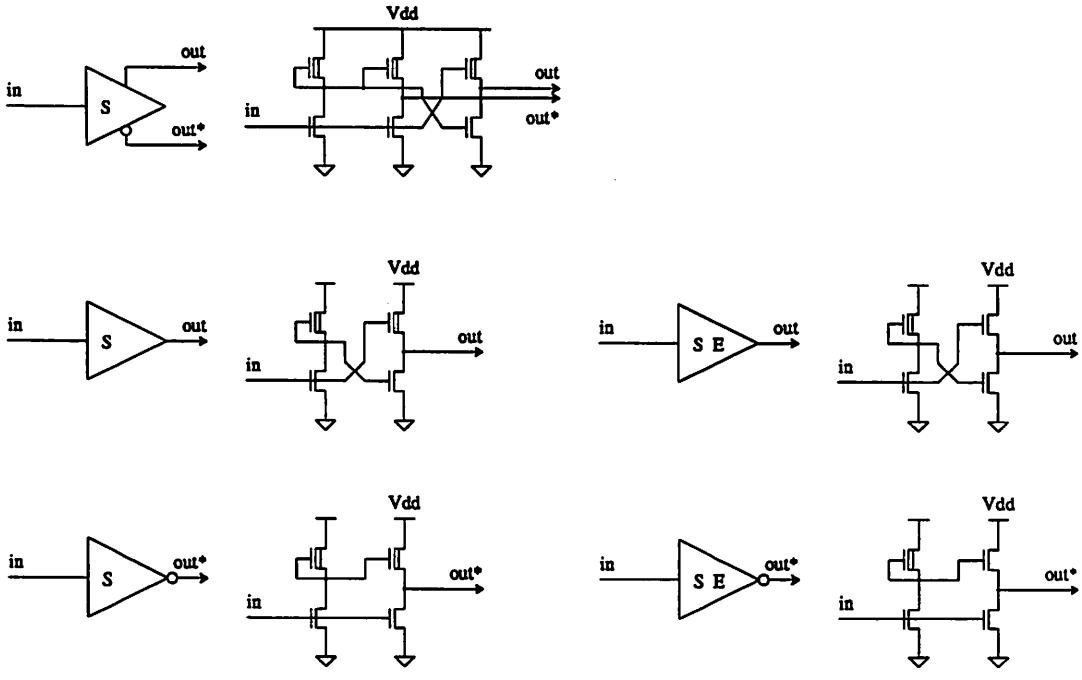



**Figure 2. Buffers**

## 6.3 THE DATA-PATH CELLS

Different data paths can make use of the same cell library because the cells have been designed with some common standards and the data path generator has some degrees of freedom when assembling the data path. All data path cells have some common characteristics which makes it possible to assemble them automatically in a fairly efficient manner. The heights of all cells in the same slice must be the same and the width of all cells for the same block must be the same. The generator can insert space between cells and slices when needed to create space for routing wires. In this way, the cells are not specialized for a particular application, they are just connected in a custom way at assembly time.

There are certain conventions that have been followed when designing these cells that are not required by the data-path generator. First, the control slice (type CNT) is at the bottom of the data path and contains the $V_{DD}$ bus and all control signals that can be brought in at the MSB side of the data path. The slices then stack from bottom to top (MSB to LSB). The top slice (GND) contains connections to ground, the clocks and any control signals that need to connect to the LSB slice.

There are basically three types of cells: storage cells, buffers and arithmetic-logical cells. The arithmetic-logical cells include: a comparator, an adder, a 2 to 1 MUX, a 2 to 2 MUX, an incrementer, an absolute value cell, inverters, zero cells and shifter cells.

There are many cells that are not really needed because the function performed by these cells can be performed by using more general cells. For example, the full adder could replace the comparator and incrementer. However, to achieve reasonably efficient space utilization, special cells are created when the space savings is significant. In the sorting filter, many comparators are used and hence space and power consumption could both be reduced by using a comparator instead of an adder. In addition, these specialized cells are then added to the cell library for use by future designers. It is always up to the data path designer whether he/she wishes to produce a better data path by customizing some cells or just to get a data path in the least amount of time possible.

### 6.3.1 Arithmetic Logic Cells

The comparator is a ripple carry type and is shown in figure 3. It has different even and odd slices

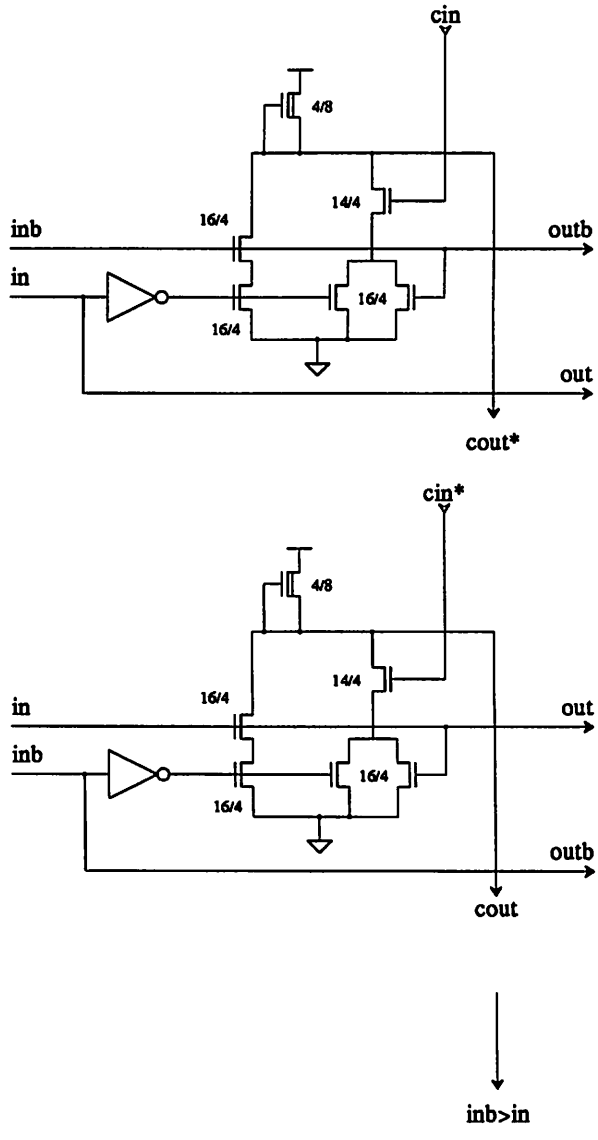



**Figure 3. "compare", EVEN slice (top), ODD slice (middle), CNT slice (bottom)**

to minimize the delay time in the carry chain by having only one gate delay per slice. Essentially this circuit is the carry chain of an adder with out the sum circuit and hence is quite a bit smaller than an adder. It is interesting to note that the two circuits are identical except that the inputs have been re-labeled. Therefore only one circuit had to be laid out by hand. Simulations showed that the carry chain requires approximately 4 nSec/slice in 4μ Nmos technology. This circuit was used in the sorting filter and the feature extractor.

The incrementer also utilizes different even and odd slices and is shown in figure 4. This circuit is essentially the same as an adder with one input always tied to ground. Incrementers are commonly used

in up counters. In the control slice, signals are generated to saturate a saturating register when appropriate.

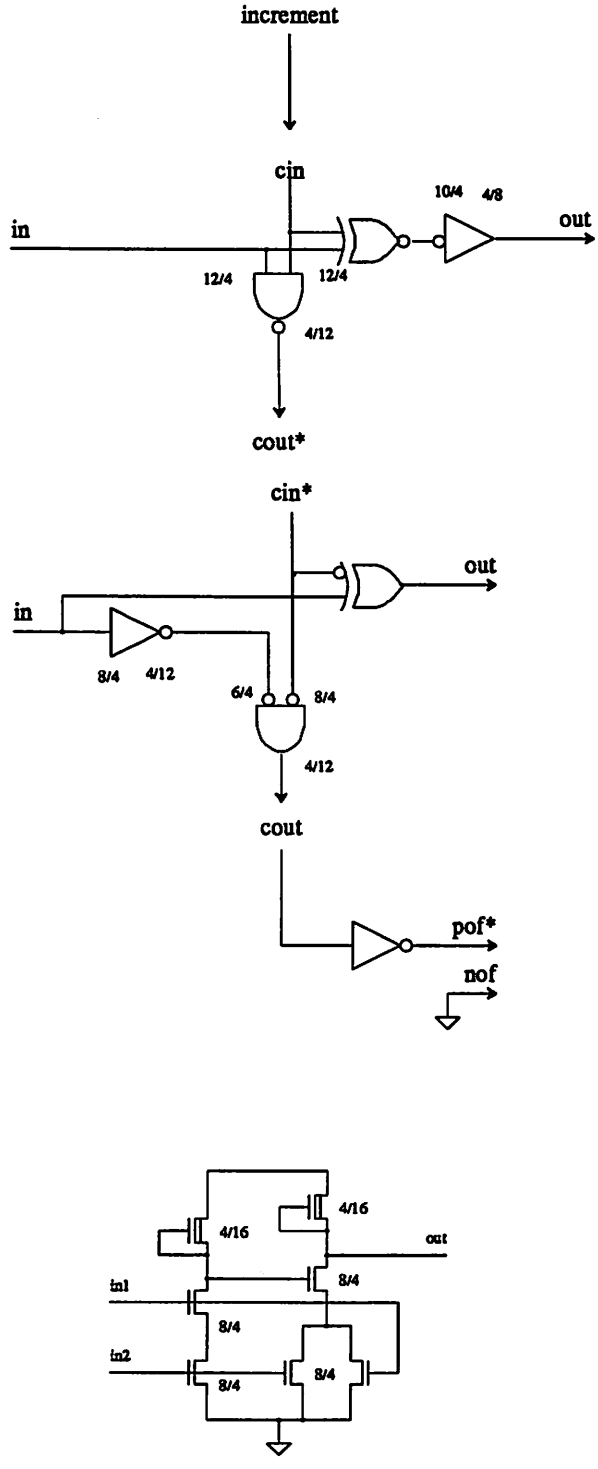



**Figure 4. "increment", GND slice (top) EVEN slice, ODD slice, CNT slice, XNOR gate (bottom)**

The full adder is shown in figure 5. The even and odd slices are different to minimize the delay path in the carry chain. In addition there are two carry inputs and two carry outputs. One input and output is in the critical path while the other input and output is in the sum path. By designing the circuit this way it was possible to keep the loading of the critical carry node to two transistor gates.

The carry input for the entire adder comes in from the GND slice to allow proper twos-compliment subtraction and other functions. The CNT slice has saturation logic (figure 6). There is a saturation circuit for both signed operands and unsigned operands.

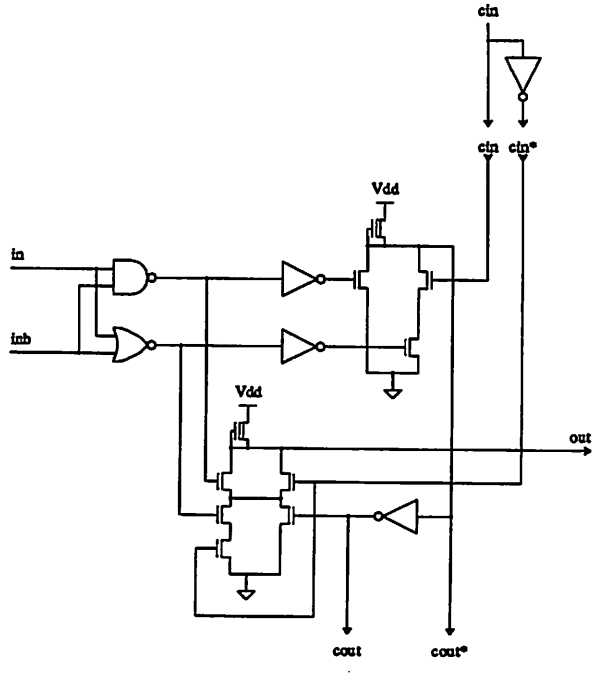



**Figure 5 a-b. "adder", "posadder", GND slice (top), EVEN slice (bottom)**

Absolute value, inversion and zeroing can all be accomplished with the "absvalue" block shown in figure 7. Basically, the circuit is an XOR gate that can be used to perform absolute value if the invert signal is derived from the data sign bit or inversion if the invert signal comes from some other source. A pull down transistor makes it possible to zero the result. The circuit generates a carry signal that can be connected to an adder carry input to make the operations true twos compliment by adding one if an inversion is performed without a zeroing. There are several simple circuits that make it possible to invert or zero the data path signal or to insert constants. The "invert-clear" block (figure 8) inverts the signal with a selectable clear. The "zero" block is just a selectable clear (same as "invert-clear" with out the

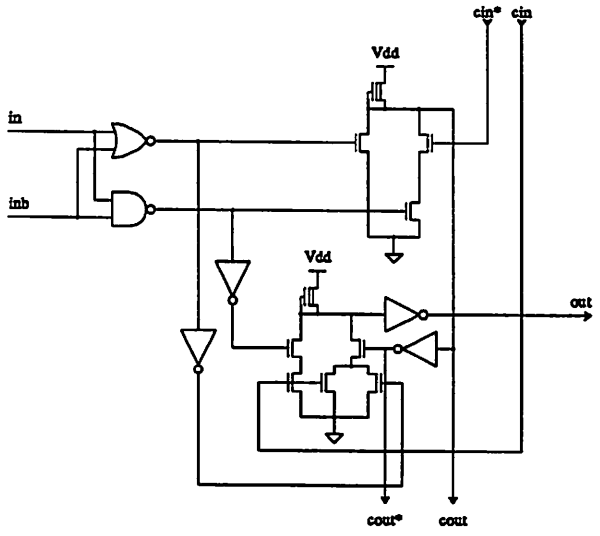



**Figure 5 c. "adder", "posadder", ODD slice**

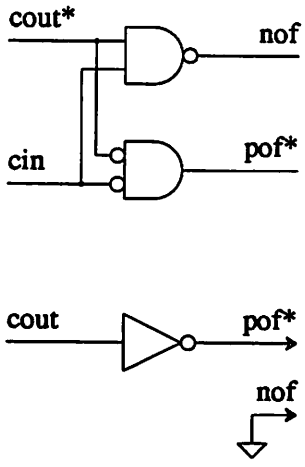



**Figure 6. "adder" CNT slice for signed operands (top)**
**"posadder" CNT slice for unsigned operands (bottom)**

inverter). "Zero-one" is identical to "invert-clear" with the input grounded. It allows the insertion of zero or one into the data path.

There are three multiplexors. A non-inverting 2 to 1 MUX (figure 9), an inverting 2 to 1 MUX with logic level restoration (figure 10) and a non-inverting 2 to 2 MUX (figure 11).

The 2 to 2 MUX is used in the sorting filters where a data swap is required. The 2 to 1 MUX with logic restoration is used in cases where the bi-directional nature of the pass transistor is not desirable.

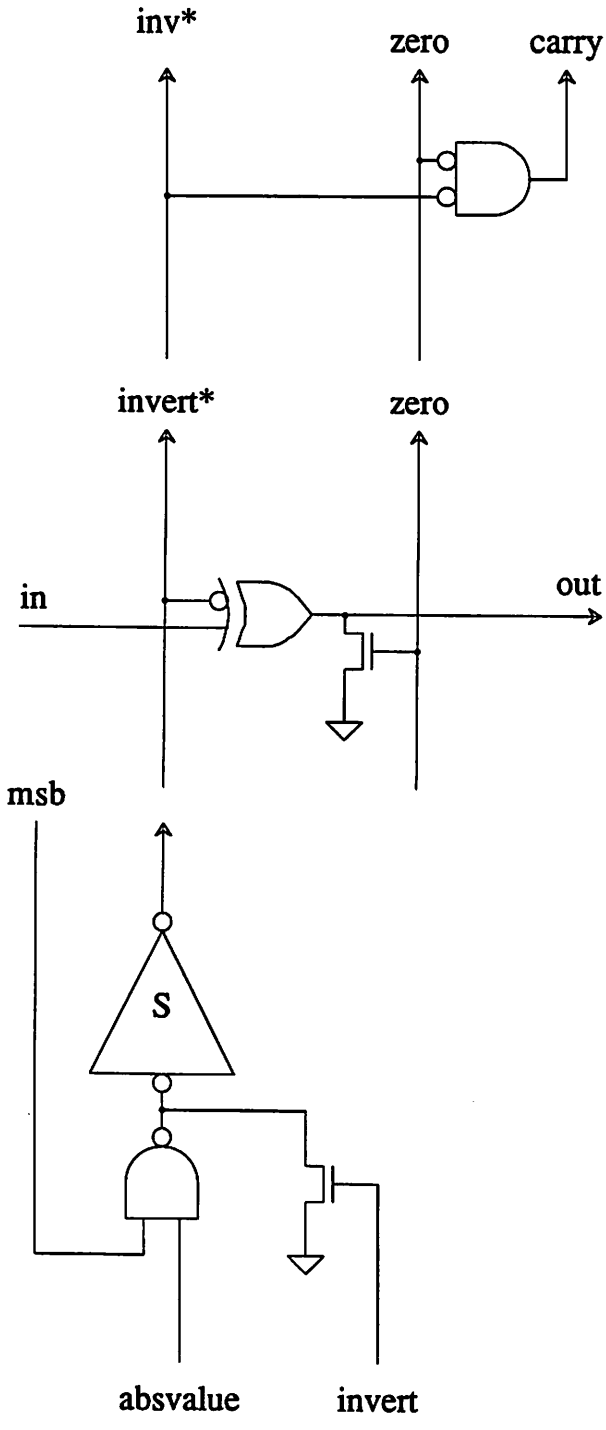



**Figure 7. "absvalue", GND slice (top), CELL slice (middle), CNT slice (bottom)**

### 6.3.2 Registers

There are two major classes or registers: static and dynamic. The static registers are more complex and are used in cases where asynchronous loading and static operation are desired. "smlatch" (figure 12)

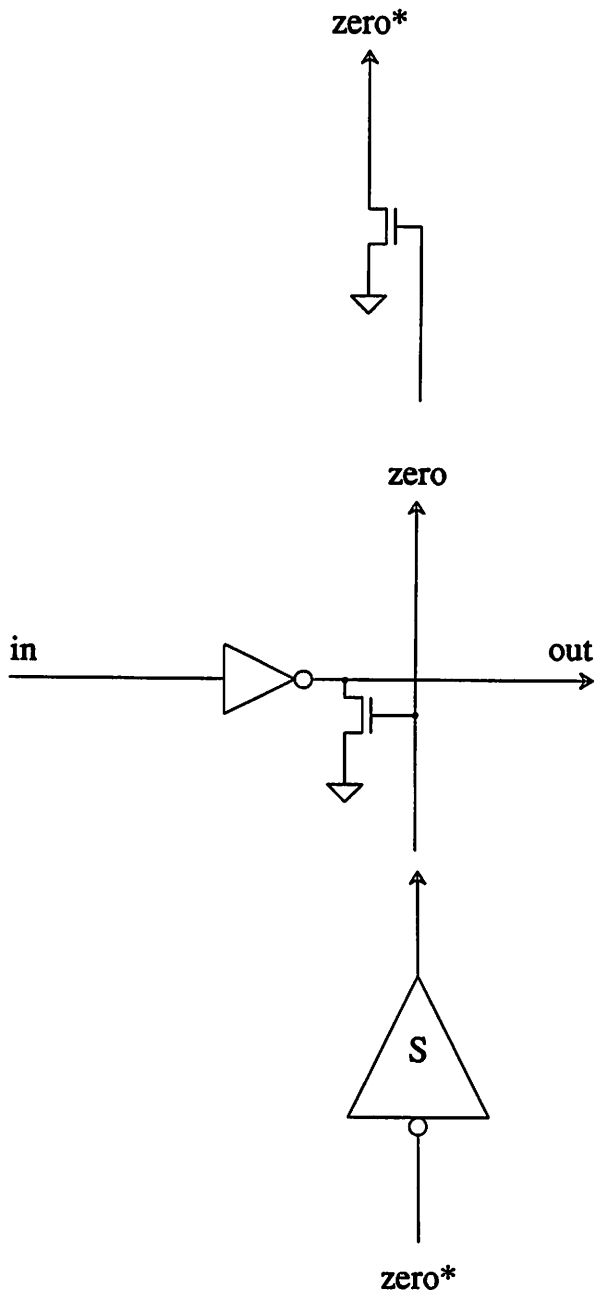



**Figure 8. "invert-clear", GND slice (top), CELL slice (middle), CNT slice (bottom)**

is a basic cross-coupled NAND gate RS latch. NAND gates were used to minimize power dissipation because only one gate is consuming power when the register is not being loaded. "latch" (figure 13) was used in the programmable arithmetic controller in the convolver and is similar to "smlatch" except for the addition of a tri-state output.

The dynamic registers fall into two classes: regular and saturating. The regular registers are

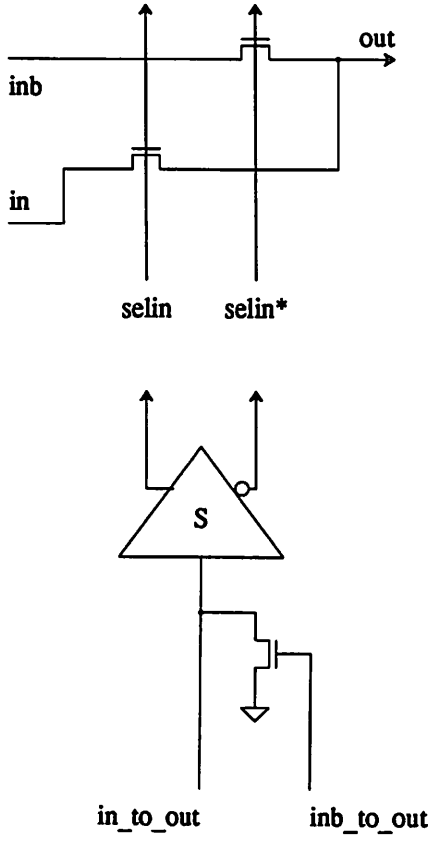



**Figure 9. "2to1mux", CELL slice (top), CNT slice (bottom)**

"delay" (figure 14), "delay1" which is the same as "delay with the two clocks reversed, "delay-clear" (figure 15) which is the same as "delay" except that is can be cleared and "delay2" (figure 16) which is a double register.

"delay" has a register in both the GND and CNT slices. The CNT slice register can be used to delay control signals (e.g. the carry input to an adder) to compensate for corresponding delays in the signal path. The register in the CNT slice is to allow the msb to be used for control purposes. In the non-linear filter post processor, the msb of a subtraction (performing a comparison) was used to control a multiplexor input. "delay2" was created for use in the sorting filter to reduce the number of routing channels that would be needed.

The are four very similar saturating registers (figure 17). "possataccum" and "possatregister" are for unsigned values and do not have a different MSB slice. "sataccum" and "satregister" are for signed values and hence have a different MSB slice. The "possataccum" and "sataccum" blocks have an output

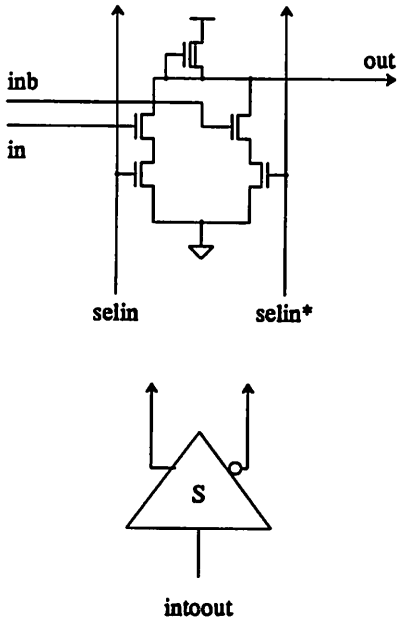



**Figure 10. "maxselect", CELL slice (top), CNT slice (bottom)**

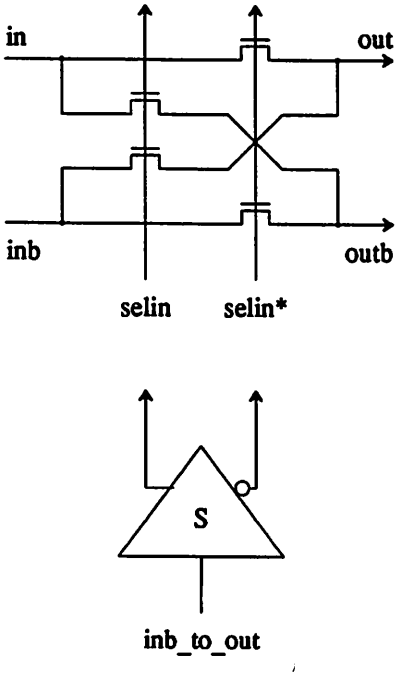



**Figure 11. "2to2mux", slice (top), cnt slice (bottom)**

connection on the left side of the cell to make accumulators easy to assemble. The register in the GND slice performs the same function that it does in "delay".

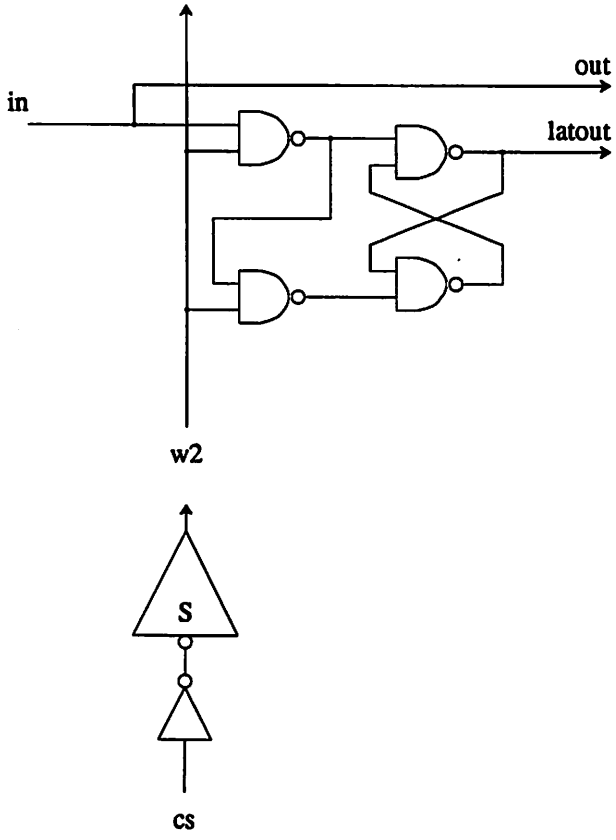



**Figure 12. "smlatch", CELL slice (top), CNT slice (bottom)**

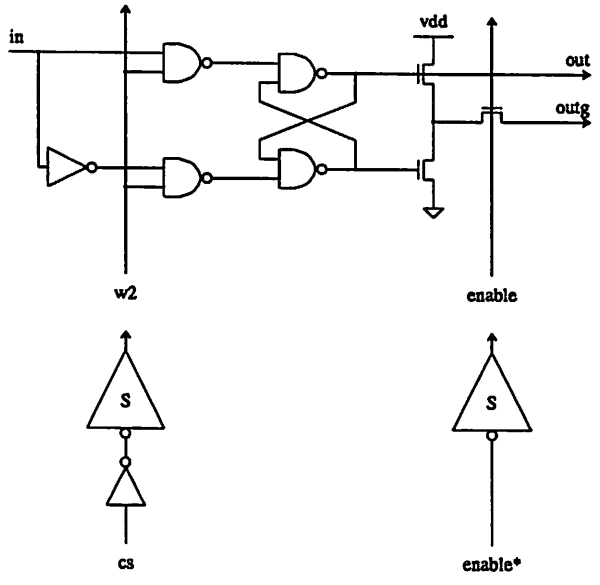



**Figure 13. "latch", CELL slice (top), CNT slice (bottom)**

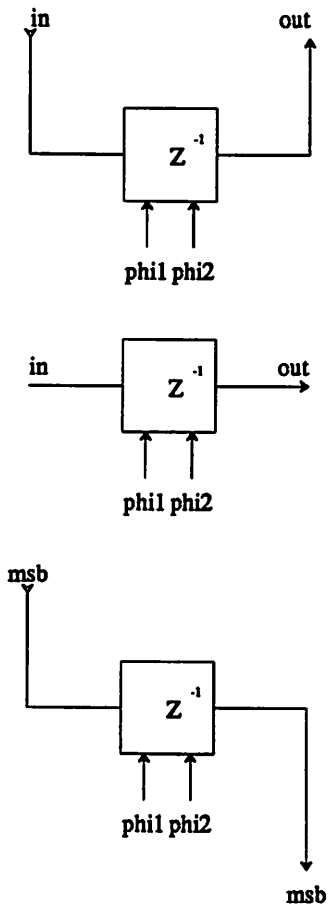



**Figure 14. "delay" , GND slice (top), CELL slice (middle), CNT slice (bottom)**

### 6.3.3 Buffers

There are basically two types of buffers: static and tri-state. The tri-state buffer, "TSbuffer", (figure 18) is used for bus driving. "buffer" is a super buffer for driving large loads (figure 19). "buffer2" is a double version of "buffer" used in the sorting filter (figure 20). "buff" (figure 21) is a smaller buffer used for logic level restoration.

## 6.4 CENTRAL STORAGE CELLS

All central storage requirements for the chips could be met with a NOR-NOR type ROM for fixed storage and the 3 transistor cell RAM for dynamic storage. For both cases, modifications were made to customize the circuit for the particular application. One reason that several versions of these circuits were designed was that it was desired to keep the core as small as possible. Therefore, general techniques like those used in the data-path compiler were considered unacceptable and custom circuits were

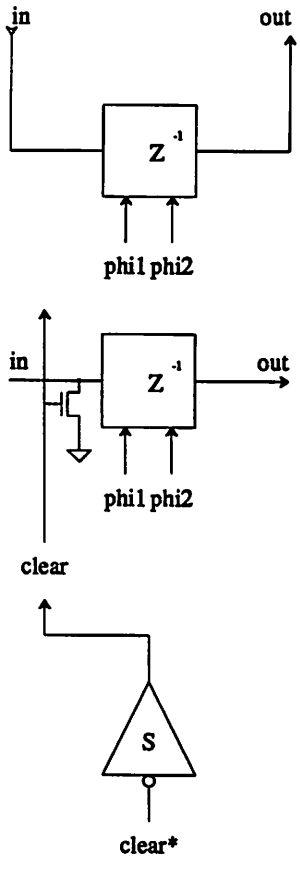



**Figure 15. "delay-clear", GND slice (top), CELL data slice (middle), CNT slice (bottom)**

designed to minimize the final circuit size. Also, because the ROM and RAM cell pitch is quite small to begin with, any increase in pitch to allow standard cells to be wired together will result in a larger percentage increase in the circuit size than in the data paths.

Three different ROMs were used. The convolver arithmetic controller had a ROM with slow row circuits but 10 MHz column circuits. The ROM look-up table chip used a pipe-lined circuit with both row and column circuits operating at a 10 MHz rate. The contour tracer FSM ROM was not pipe-lined because the circuit would not allow the extra pipe-line delays.

The basic NOR-NOR ROM configuration is shown in figure 22. This is the pipe-lined circuit used in the look-up table ROMs. Note that there is a pipe-line register at the inputs (column and row address inputs) and at the output. The row decoder inputs are driven with a standard depletion mode output driver. Because the rows have potentially large capacitance (the gates of many ROM cells) it was also desired to drive the row selects with a fast super buffer. A standard super buffer was not used because in

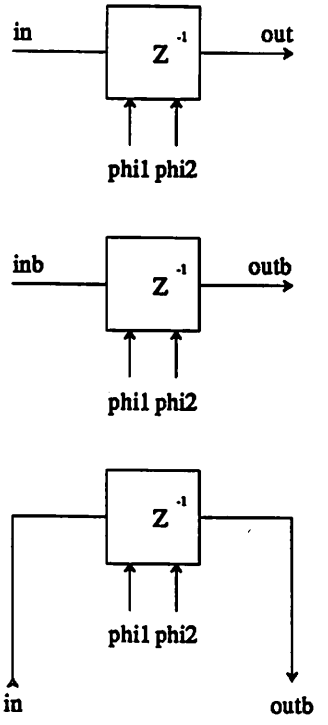



**Figure 16. "delay2", CELL slice (top), CNT slice (bottom)**

a NOR-NOR ROM all row selects except one are low at a given time. This means that all super buffers except one will be consuming power. By replacing the depletion output pull up device with an enhancement device, the buffer becomes a true push-pull circuit and consumes no static DC power (actually, the smaller input inverter will continue to consume power). The degraded high level output voltage of the row driver reduces the performance of the circuit somewhat but is still acceptable because the ROM cell has voltage gain.

The bit line has both an active and passive pull up. The active pull up quickly precharges the bit line high while the passive pull up provides a small current to offset leakage on the bit line.

To make the layout of the output register simpler, the circuit always has at least one column decode. This doubles the pitch available for the output register. After the first column decoding device, the first half of the output register latches the data. The remaining part of the tree column decoder is between the two halves of the output register. The first half of the output register isolates the bit lines from charge sharing in the last part of the tree decoder.

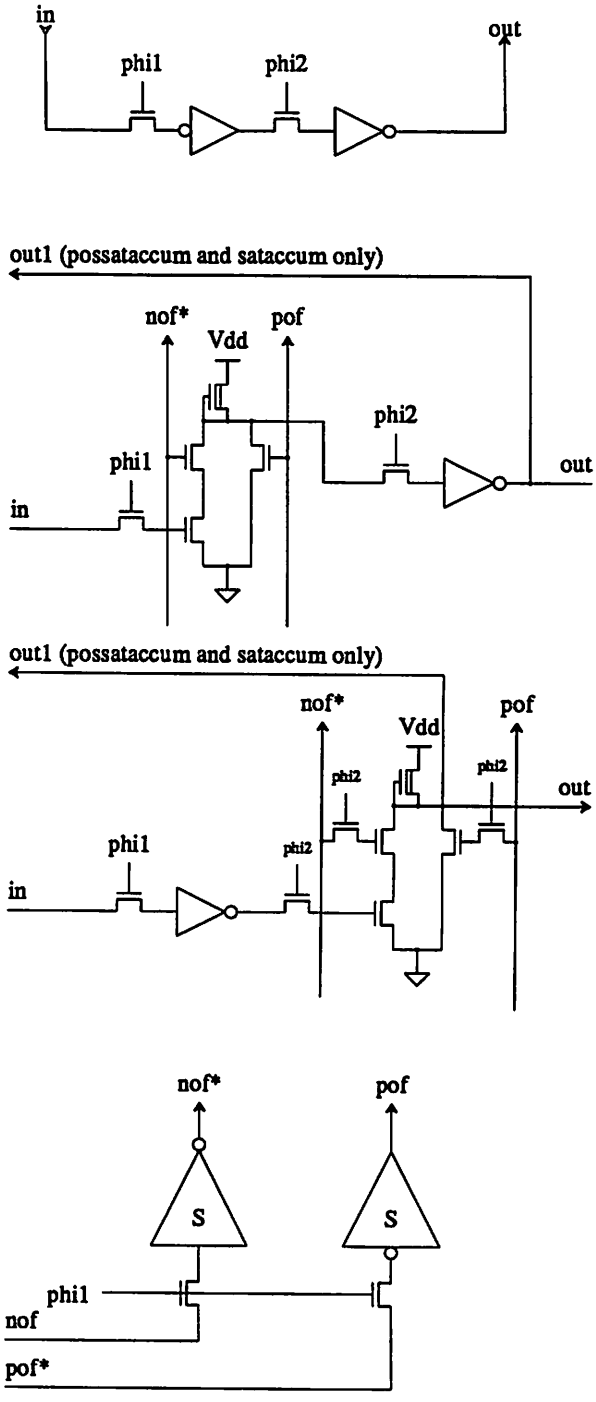



**Figure 17. "sataccum", "satregister", "possataccum", "possataccum" GND slice (top), CELL slice, MSB slice, CNT slice (bottom)**

The contour tracer FSM ROM is very similar to figure 22 except that the input registers and some of the output registers were removed (the FSM state registers were left). Further, each bit line was isolated from the column decoder because the column address may become valid after the rest of the ROM

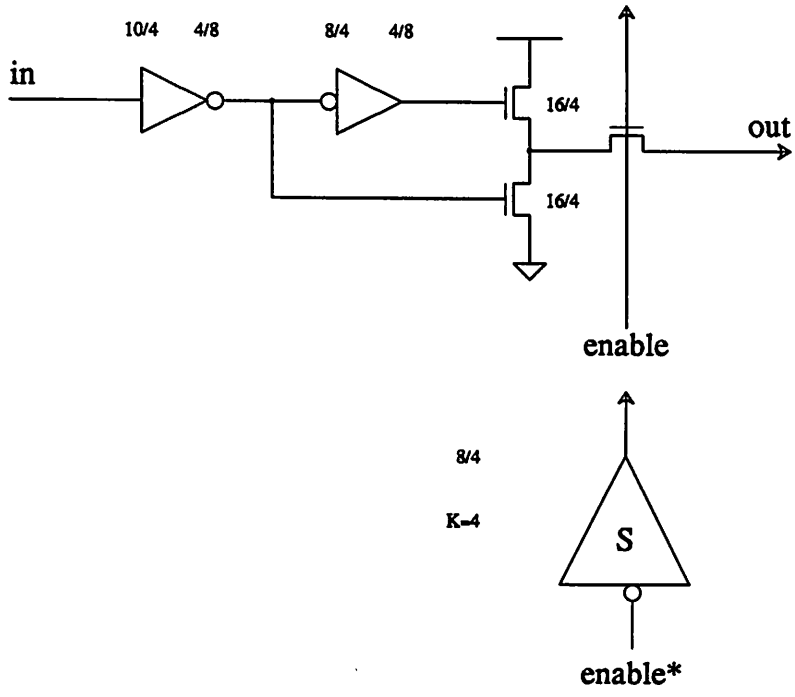



**Figure 18. "TSbuffer", CELL slice (top), CNT slice (bottom)**

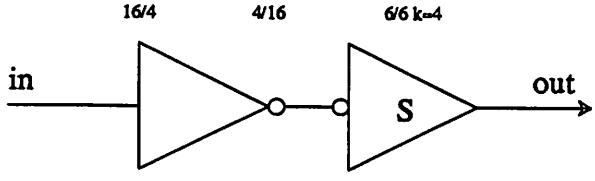



**Figure 19. "buffer", CELL slice**

settles and it was desired to prevent any charge sharing between adjacent bit lines when the column addresses change.

The convolver arithmetic controller ROM did not require registers at the input or fast row drivers because the row addresses change once at reset. The active bit-line pull-up was also not used in this particular circuit.

The line delay, contour tracer RAM, look-up table RAM and histogrammer all use the same 3T cell with different peripheral circuits.

The line delay (see section 7.8) utilizes a shift register pointer instead of more general decoding logic found in the other RAM circuits. In figure 23 the row select logic is shown. The shift register

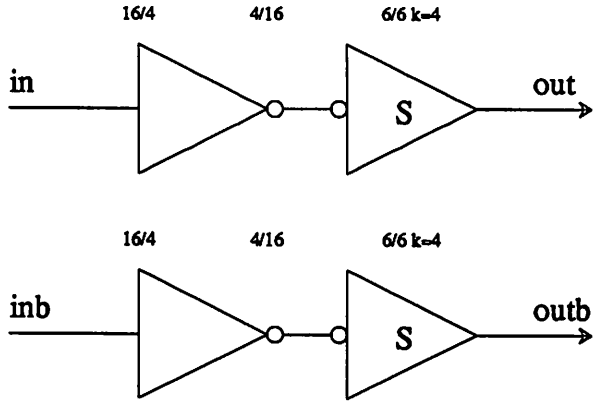



**Figure 20. "buffer2", CELL slice**

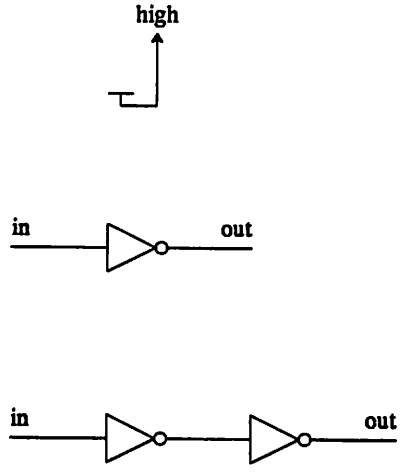



**Figure 21. "minus1", GND slice (top), CELL slice (middle), "buff", CELL slice (bottom)**

pointer and distributed OR gate (pull down devices are at all but the last two shift register outputs) keep a single row selected without the need for a reset circuit. The read row selects are taken directly from the shift register outputs. The shift register outputs are gated with "wen" to generate the write selects. Notice that the row previously read is the current row being written.

The output circuit is shown in figure 24. The circuit is basically a precharge device, load-hold register and tri-state buffer combination. The size of the circuit was reduced by using the inverter in the register to generate the true and inverted data signals needed in the tri-state buffer.

The input circuit (not shown) is just a cascade of load-hold registers. The peripheral circuits for the contour tracer RAM are quite different and were designed virtually from scratch. The input circuit

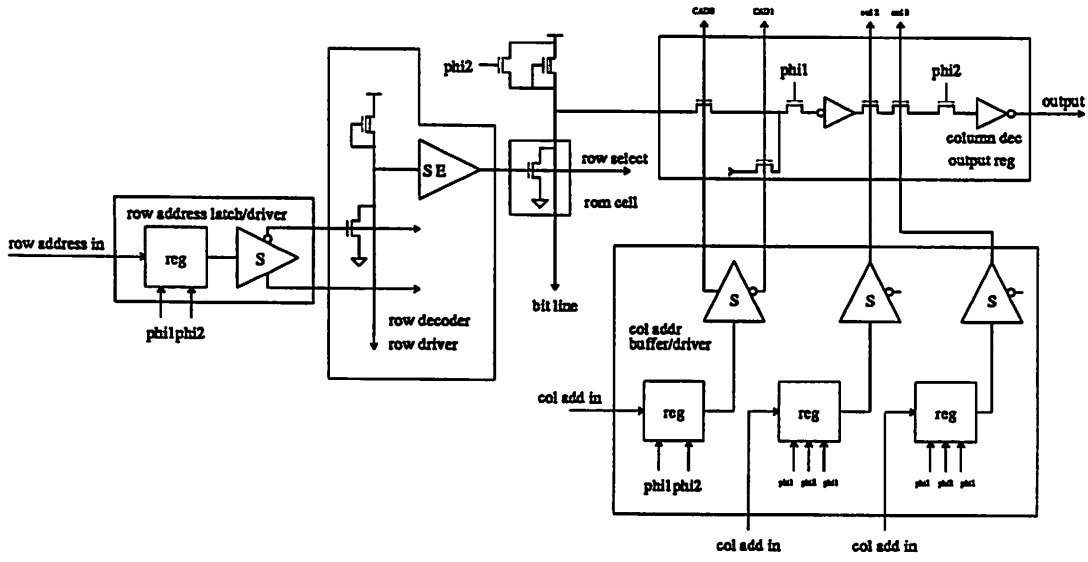



**Figure 22. NOR-NOR ROM used in the look-up table chips**

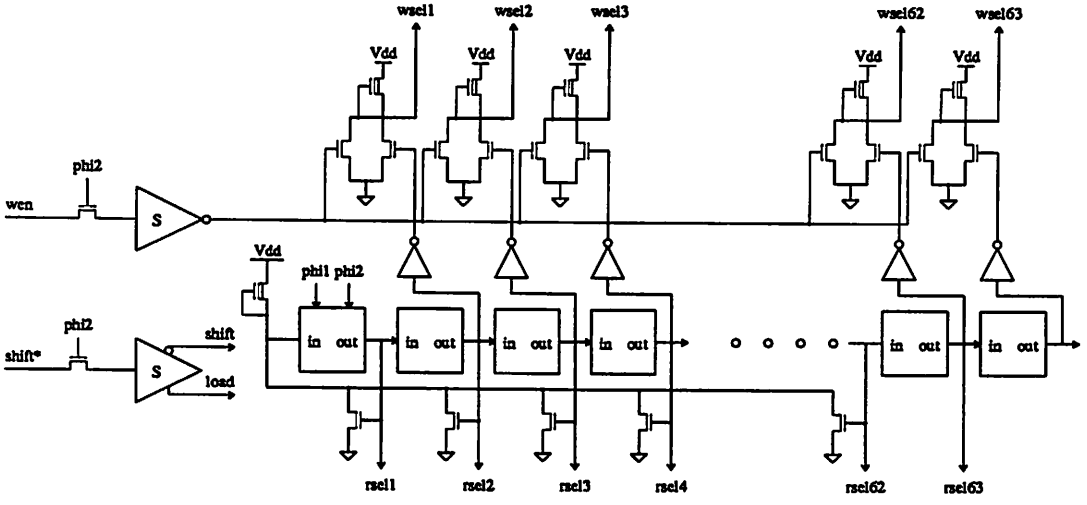



**Figure 23. Line Delay Select Logic**

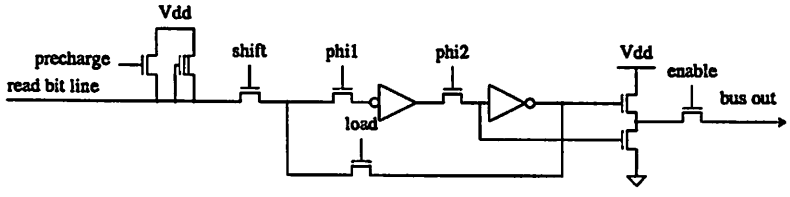



**Figure 24. Line Delay Output Circuit**

(figure 25) contains the shift register (a load-hold register) that handles the loading of one entire line of the video image in the acquire mode, a register to make perform refreshing (the data is always written the cycle after it was read) and the multiplexor that selects the proper data for the bit line based on the mode of operation.

The output circuit (figure 26) is very similar to the ROM output circuit. The bit lines are first buffered (to prevent charge sharing). The desired output is selected by a seven stage column tree decoder. The tree decoder is broken at the half way point to decrease the critical path length. For this case half of the nodes fall during a transition instead of all nodes rising, yielding a faster circuit. Further, if the column addresses are valid during precharge, the second half of the column decoder will be precharged high. When the precharge cycle is over, the column decoder circuit can be quickly pulled low when the bit line is pulled down.

The row select logic is shown in figure 27. The read select line is driven by an enhancement super buffer to minimize power consumption. The write select is simply the row select delayed by one cycle in the trace mode (for refresh). During loading, the pipe-line stage is bypassed. A standard NOR decoder is used to generate the un-buffered row selects.

The circuits for the look-up table chip and the histogrammer are similar but have modifications necessary for each particular application.

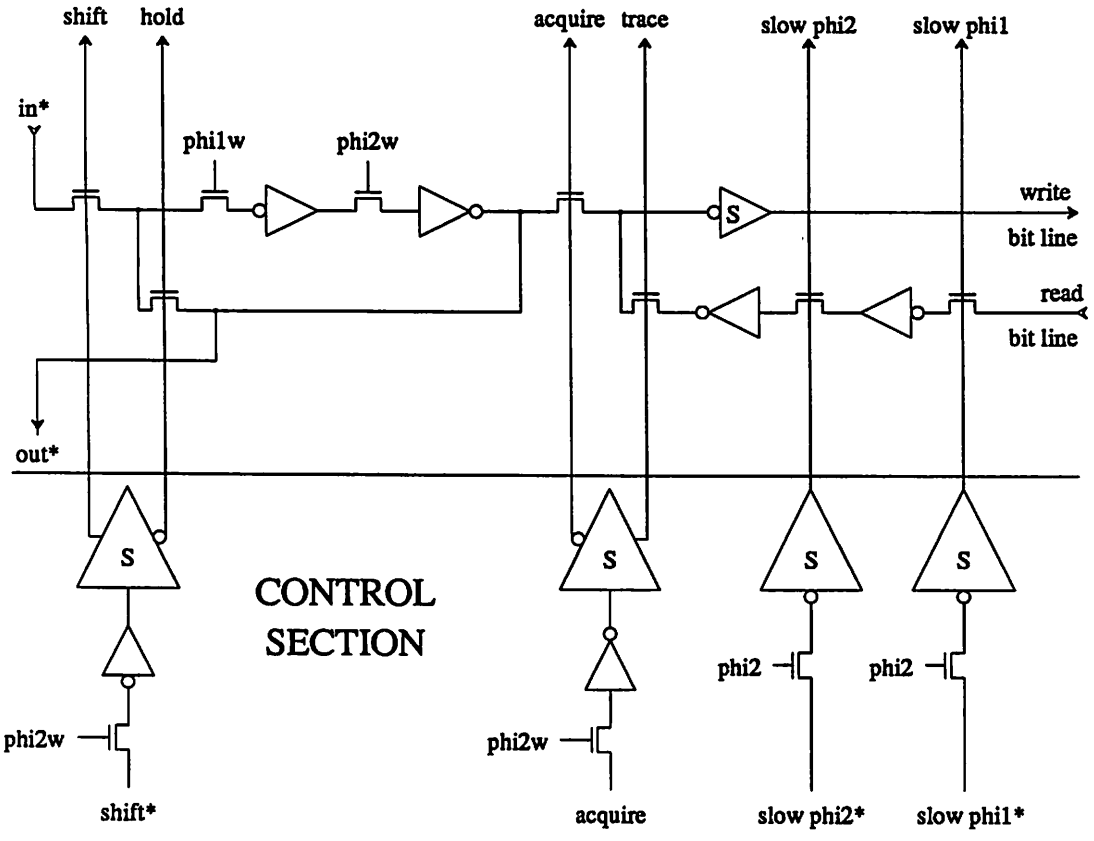



**Figure 25. Tracer RAM input circuit**

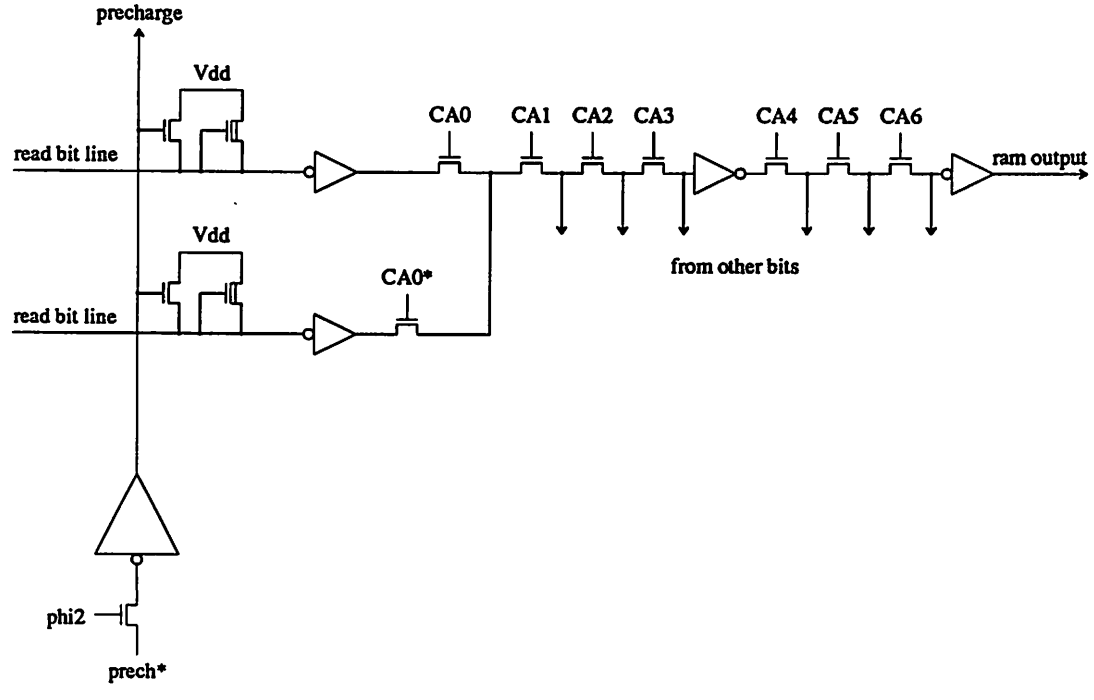



**Figure 26. Tracer RAM output circuit**

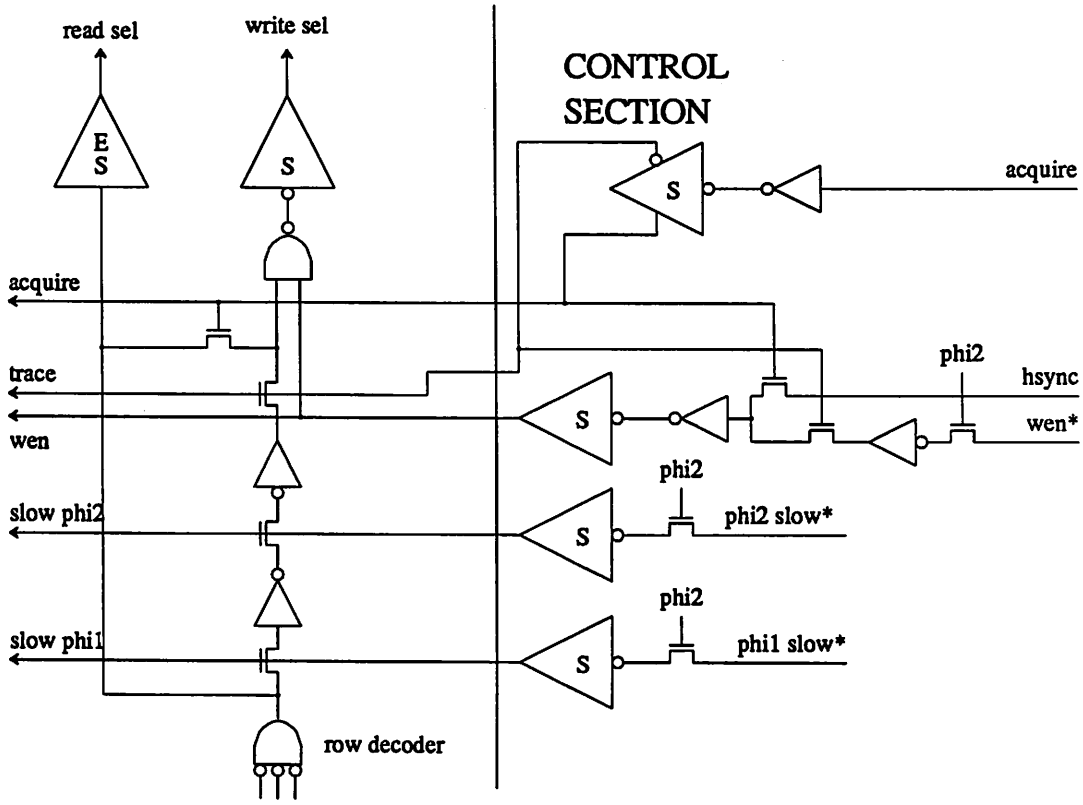



**Figure 27. Tracer RAM select Circuit**

## 6.5 REFERENCES

[1] Mead, C., Conway, L., *VLSI Systems*, Addison Wesley, Reading, Mass., 1980.

# CHAPTER VII THE IMAGE PROCESSING ICs

## 7.1 SUMMARY OF CHIPS

A set of chips has been developed to perform the functions needed for the image recognition system. The set includes linear convolvers, linear convolver post processors, a logical convolver, look-up-table ROMs, a sorting filter, contour tracers and a feature extractor.

Table 1 gives a summary of the circuit information. Figures 1 through 9 are die photos for the chips that have been produced. Note that the die photos are not uniformly scaled.

### 7.1.1 The Linear Convolvers

Three different 3x3 linear convolvers were developed to perform edge extraction and noise rejection functions. These include a convolver for 512 pixel lines with impulse responses stored in ROM (figure 1), a convolver for 512 pixel lines with a programmable impulse response (figure 2) and a 256 pixel line convolver with ROM based impulse responses (figure 3).

### 7.1.2 Linear Convolver Post Processors

2 chips were developed to perform additional operations on the outputs of the linear convolvers. One performs the absolute value, sum and threshold required in the Sobel filter while the other performs the additional operations required to compute the "out of bounds" filter output (figure 4). Both of these chips are fairly small data paths.

### 7.1.3 The 7x7 logical Convolver

The 7x7 logical convolver is used to perform binary filtering in general and edge dilation in the image recognition system. Only one version for 512 pixel lines has been developed (figure 5).

### 7.1.4 3x3 Sorting Filter

This circuit computes the maximum, minimum, median or maximum difference over a 3x3 region. The median is used for noise rejection while the maximum difference can be used for high pass filtering for edge extraction. Again, only one circuit for 512 pixel lines was developed (figure 6).

### 7.1.5 Contour Tracer

Contour tracers for 512 and 256 pixel lines were developed (figure 7). The contour tracer scans for images in the edge map. When one is found, the image is traced and the curvature is output. To make it possible to put the edge map RAM on a single 4μ NMOS chip, the edge map is down-sampled to 128x128. Therefore, the two tracers only require differences in the down-sampling logic and not in the amount of storage required.

### 7.1.6 Feature Extractor

The feature extractor computes features (bounding box, center of mass, area and perimeter) from the curvature signal output by the contour tracer (figure 8). It also provides the interface between the tracer and the host computer.

### 7.1.7 Look up table ROM

These circuits can be programmed to store an arbitrary memory-less transfer function for 8 bit gray values. Linear to logarithmic and linear to exponential tables have been produced (figure 9).

| circuit | line length (pixels) | area (sq. mm) | number xsters | Ops/sec* (Millions) |
|---|---|---|---|---|
| 3x3 convolver | 512 | 42 | 39K | 280 |
| 3x3 convolver | 256 | 35 | 25K | 140 |
| 3x3 sorting filter | 512 | 42 | 35K | 410 |
| 7x7 logical convolver | 512 | 18 | 16K | 1470 |
| contour tracer | 512 | 45 | 57K | 12 |
| contour tracer | 256 | 45 | 57K | 12 |
| look-up-table | NA | 2** | 2K | 10 |
| Sobel post processor | NA | 1.5** | 1.3K | 50 |
| non-linear LPF processor | NA | 2** | 1.5K | 60 |
| feature extractor | NA | 30 | 7.5K | 40 |

* includes: memory accesses, arithmetic and logical operations.

** active area

**Table 1. Circuit Information**

Figure 1. 3x3 Convolver for 512 pixel Lines, ROM Based Coefficients

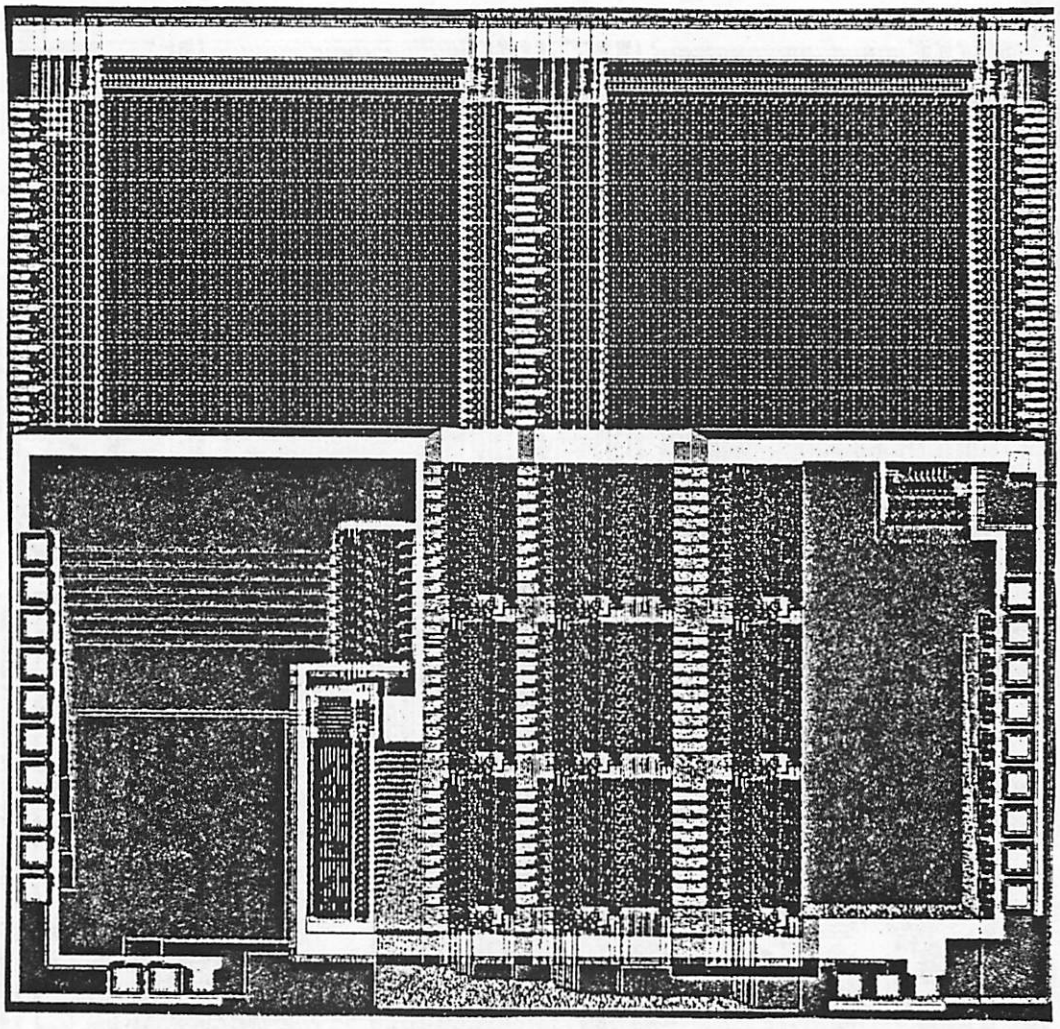


**Figure 2. 3x3 Convolver for 512 pixel Lines, Programmable Coefficients**

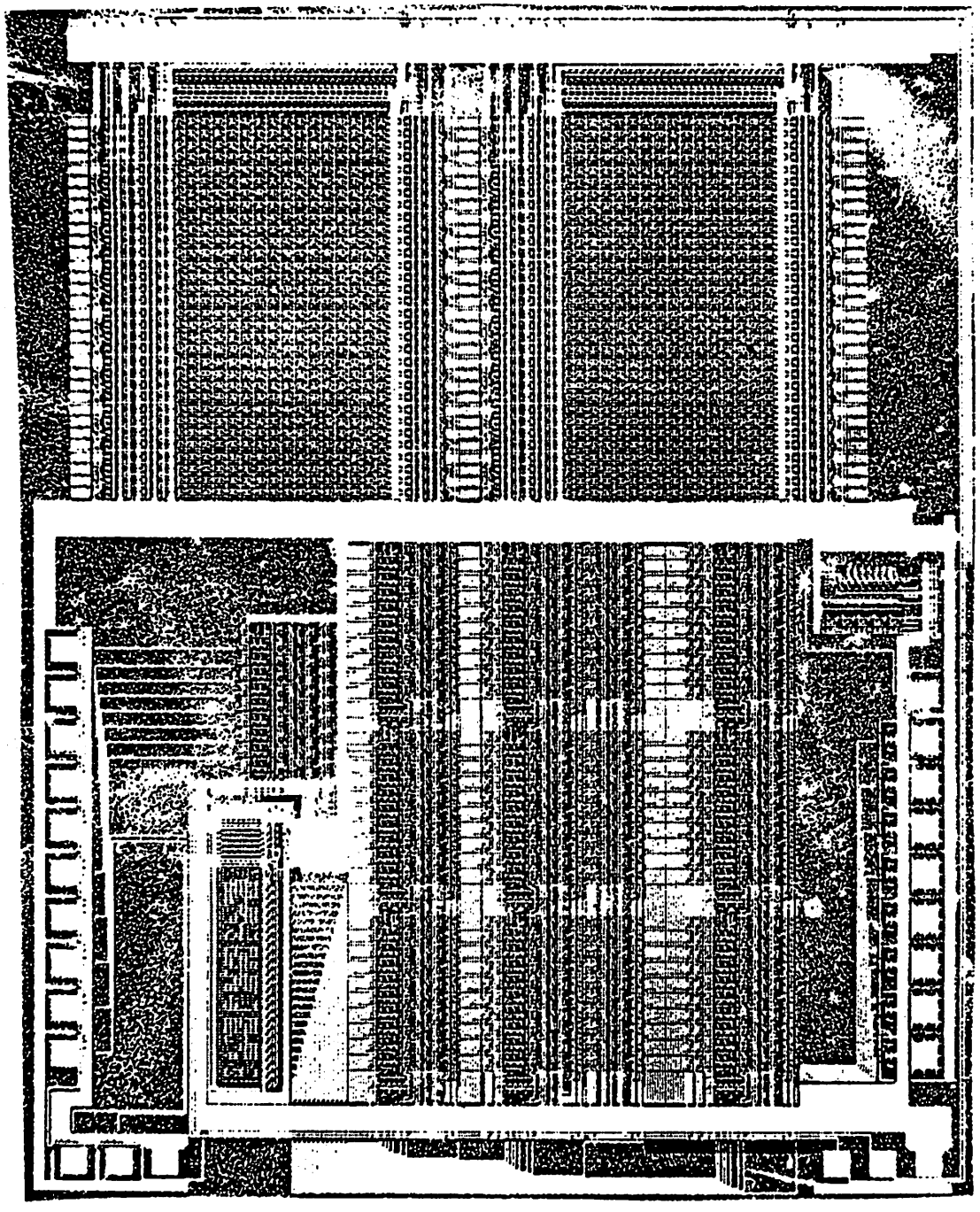


**Figure 3. 3x3 Convolver for 256 pixel Lines, ROM Based Coefficients**

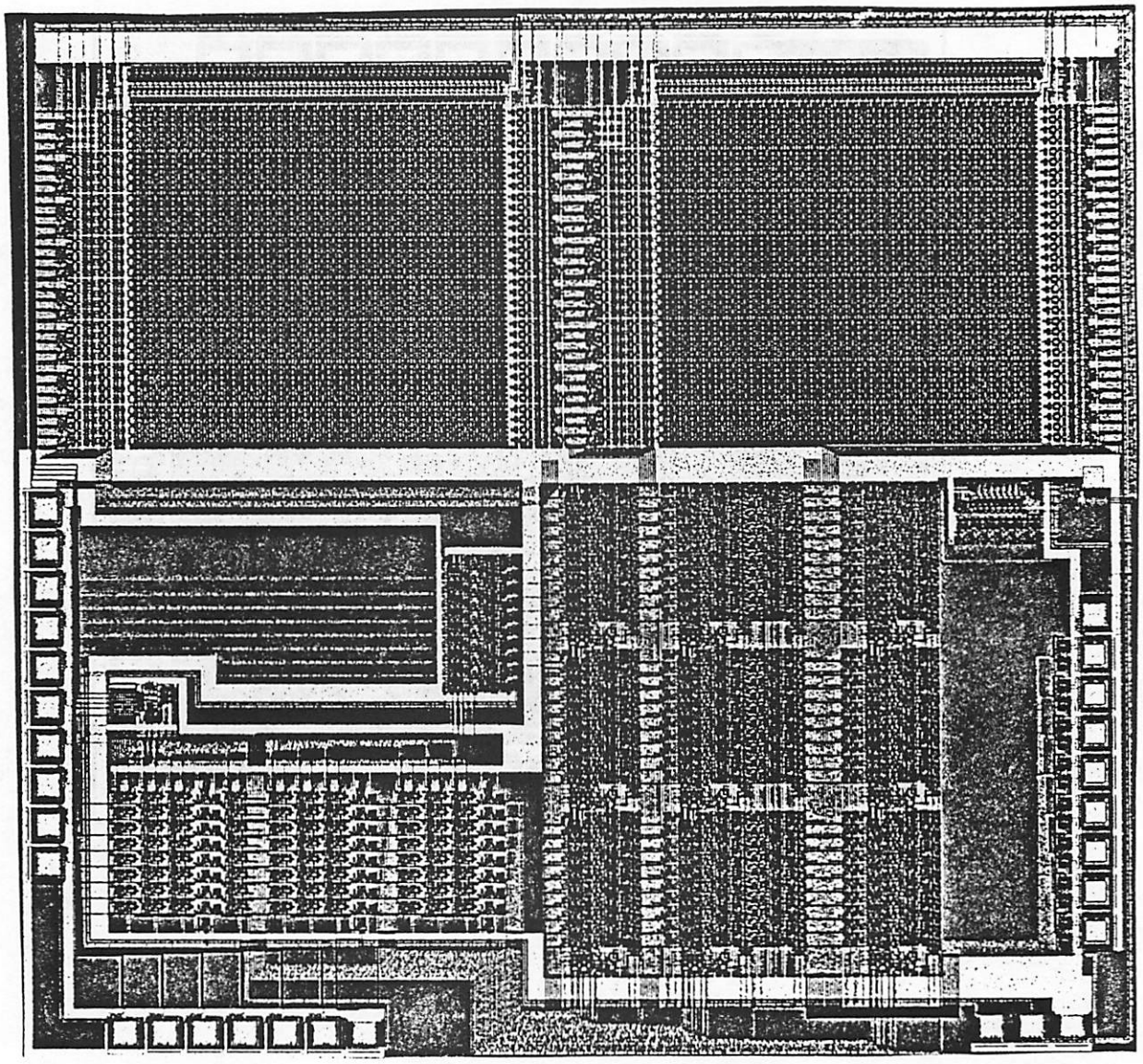


Figure 4. Non-Linear Low-Pass Filter Post Processor

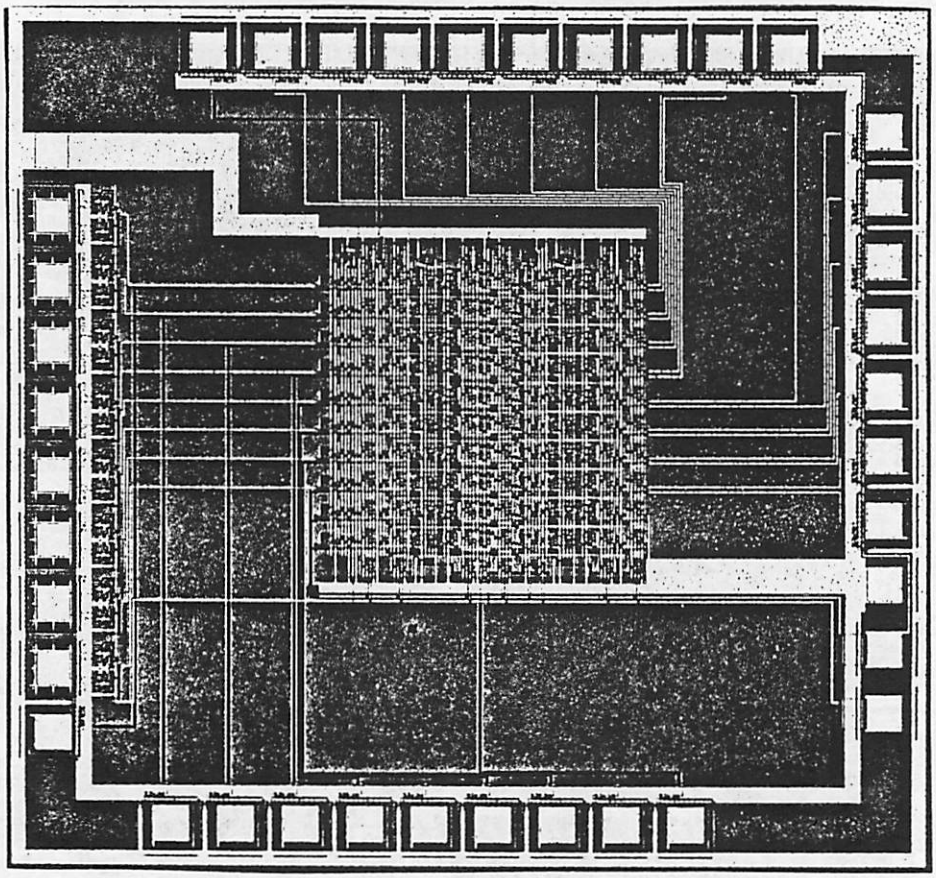


**Figure 5. 7x7 Logical Convolver (512 pixel lines)**

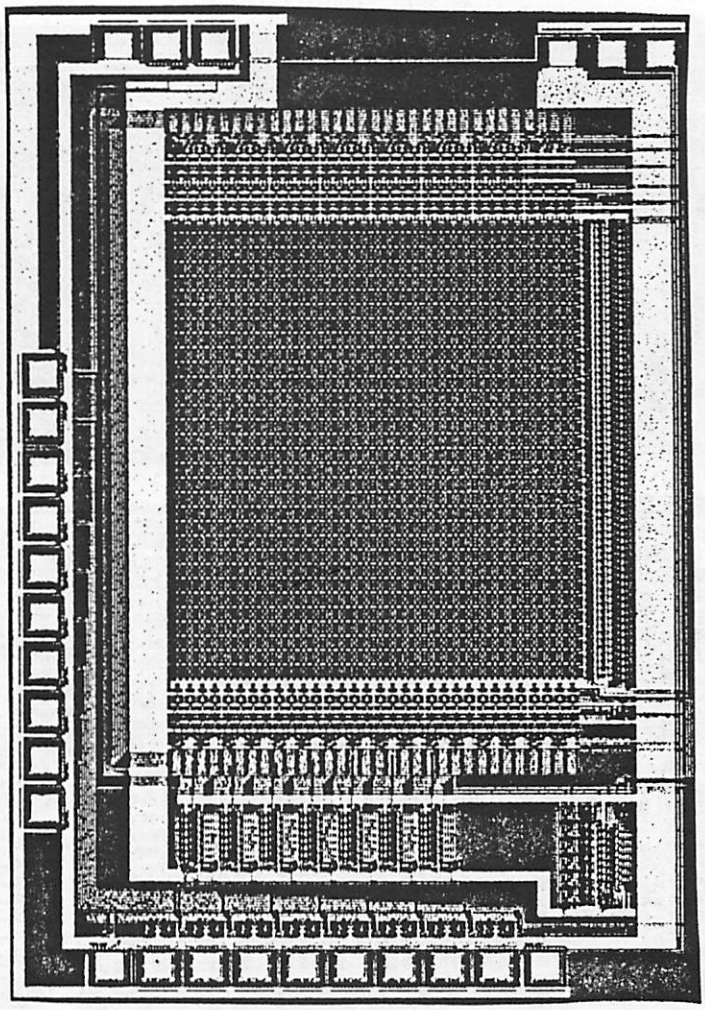


Figure 6. 3x3 Sorting Filter

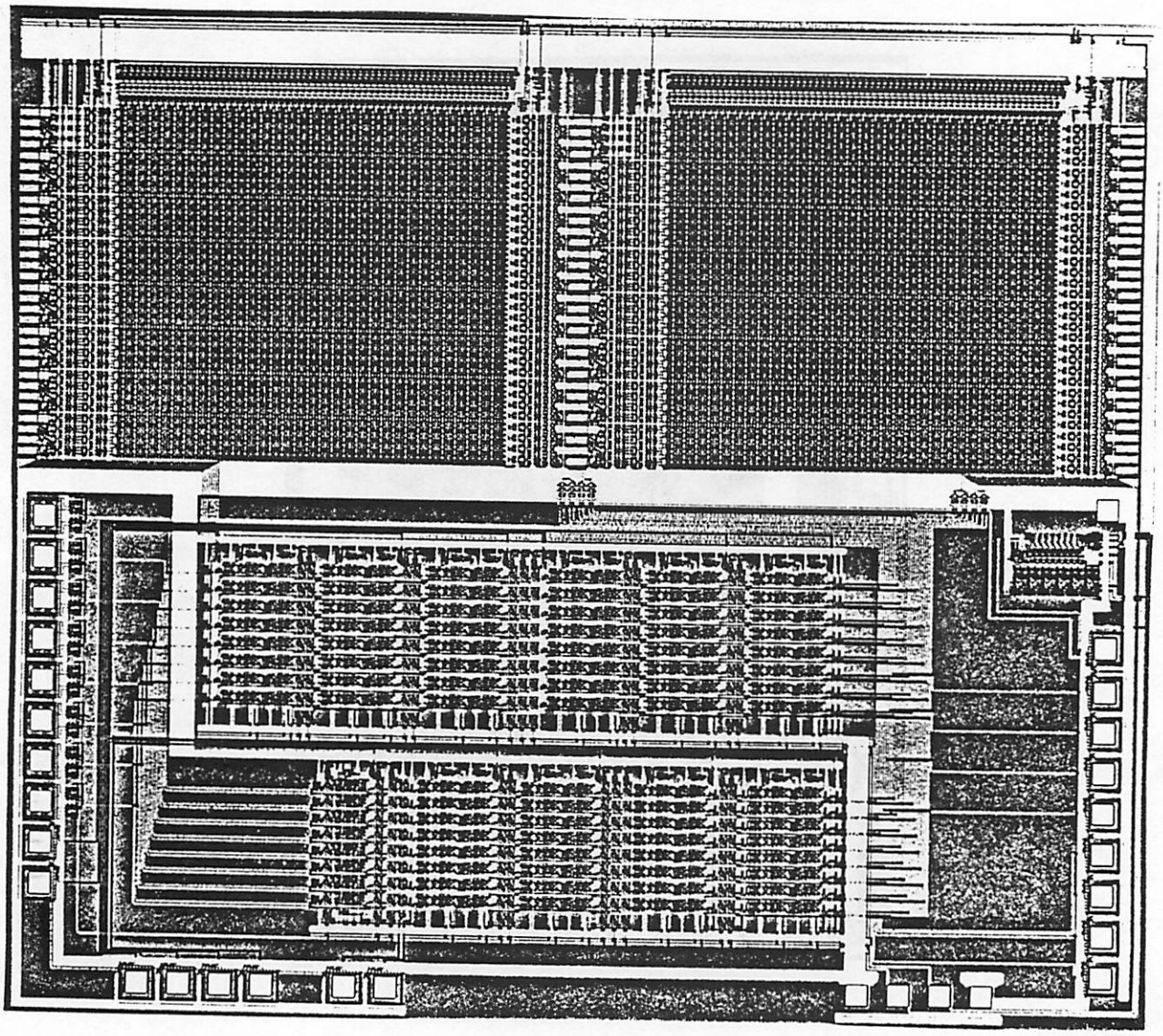


Figure 7. Contour Tracer

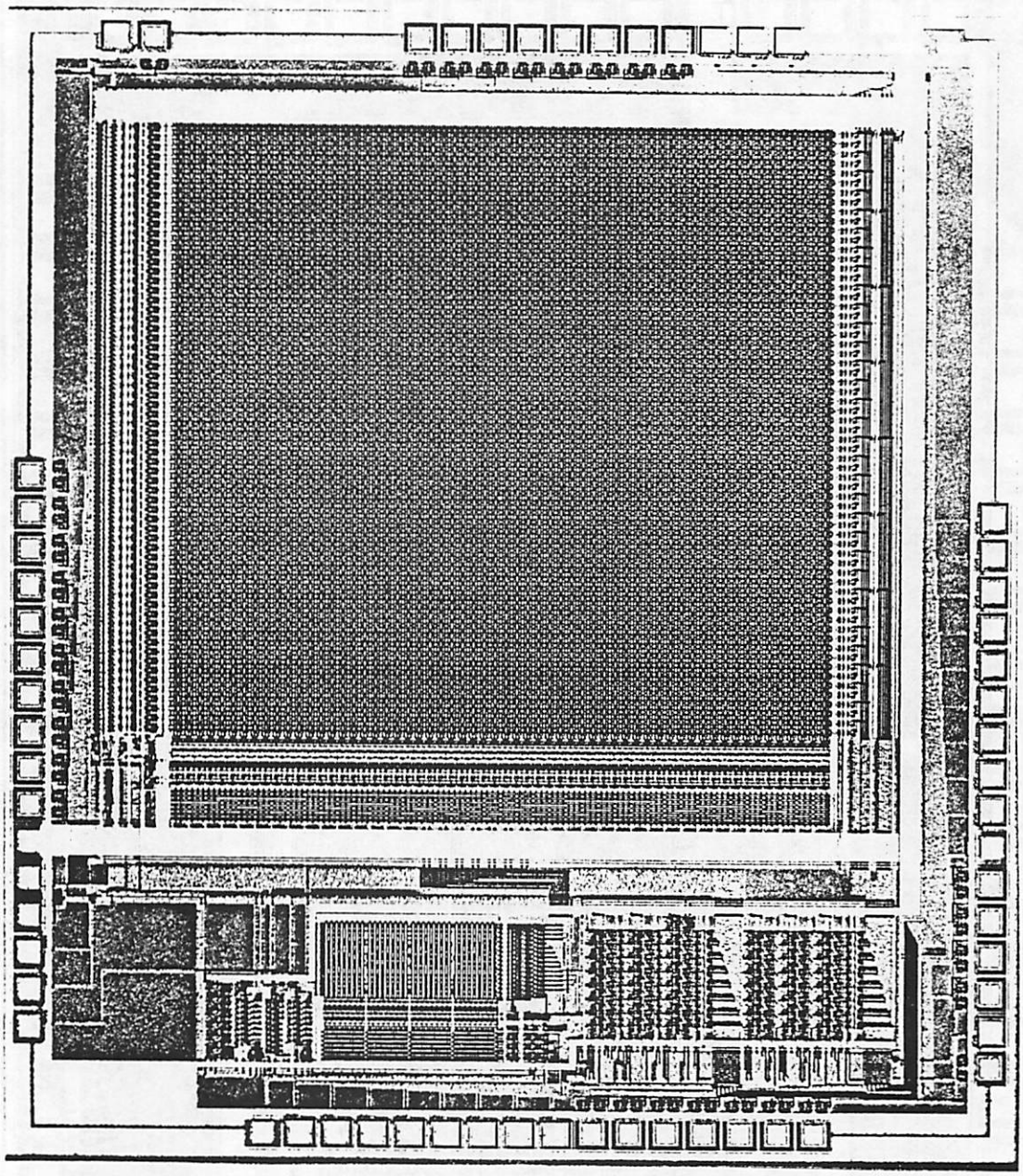


**Figure 8. Feature Extractor**

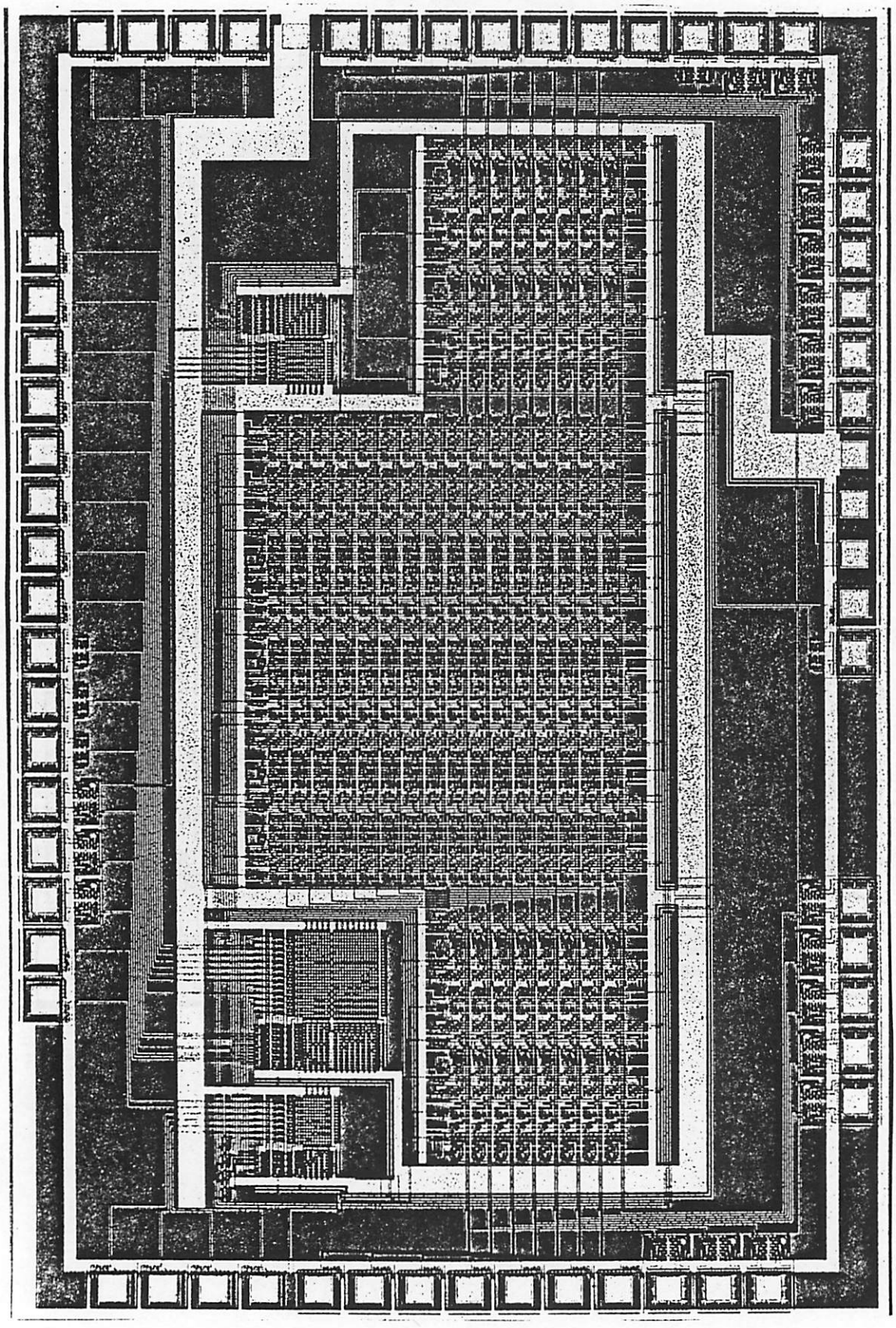
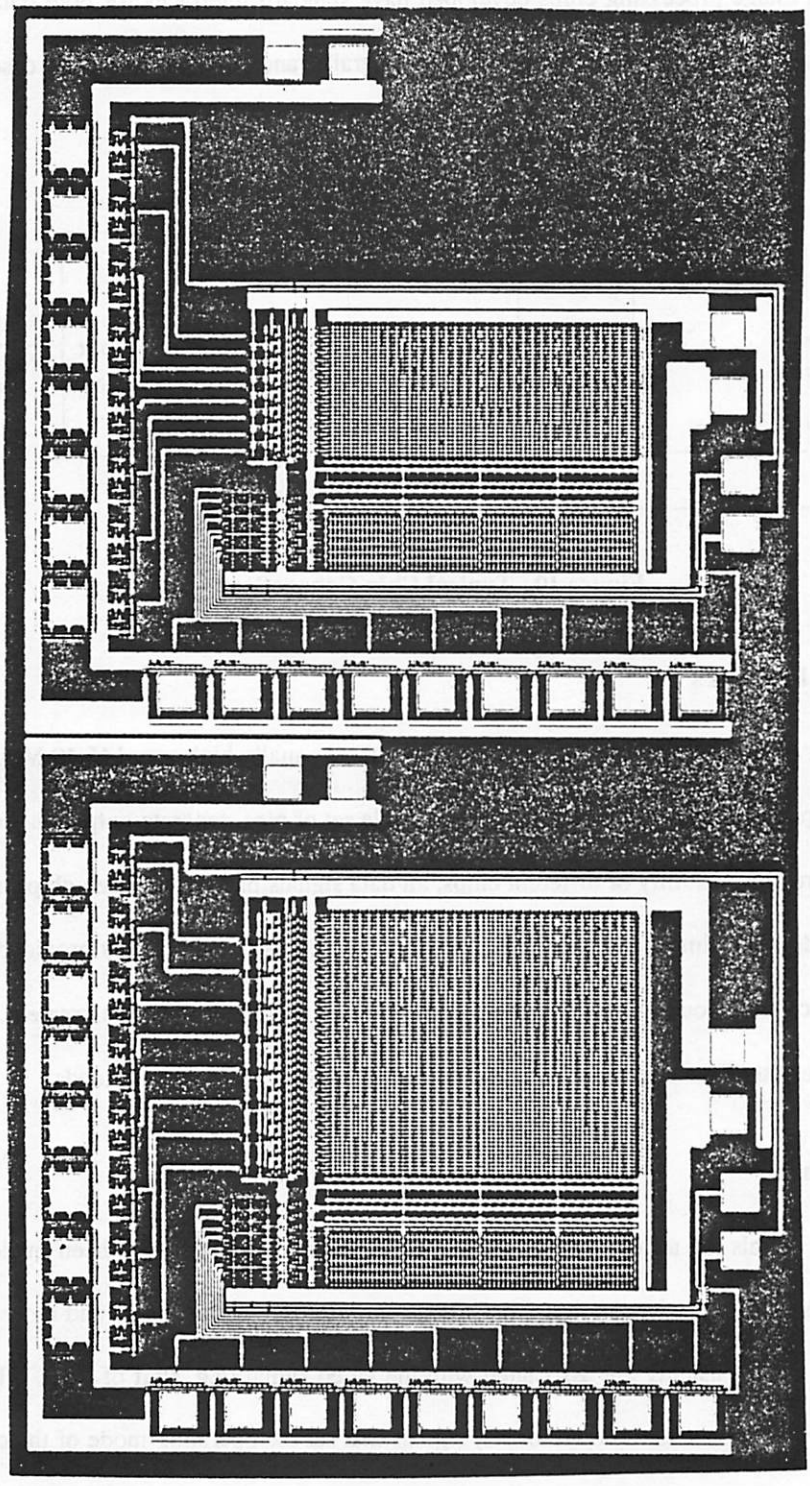


**Figure 9. LUT ROMs**



## 7.2 IMAGE PROCESSOR INTERFACES

Most of the image processing chips developed have standard interfaces for both data and control signals. In this way, circuits can be easily connected in parallel and cascade. A typical cascade connection of chips is shown in figure 10.

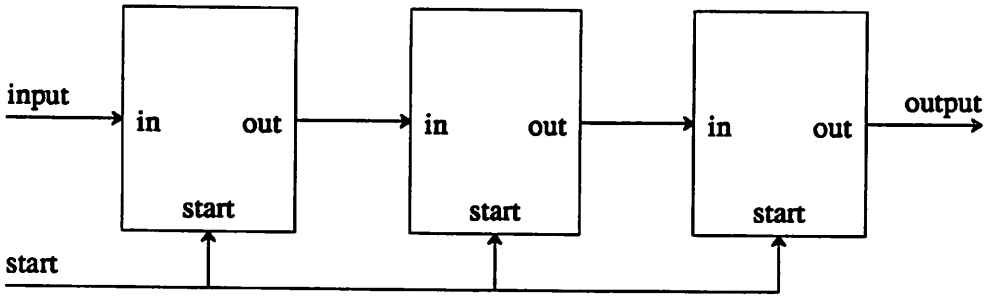



**Figure 10. Typical Chip Connection**

### 7.2.1 Data Signal Interfaces

Data signals are the input or output video signals and are usually high speed (5-10 MHz). To avoid the need for multiplexing the inputs and outputs on a single set of pins, separate output and input pins are provided. To ensure compatibility of different chips, all data signals passing between chips are valid during ph2. To provide more timing margin when going between chips, a ph2 latch is provided in the input pads and a ph1 latch in the output pads (standard pads with the registers have been created). In this way, an extra pipeline register is inserted to accommodate delays when driving off-chip loads.

### 7.2.2 Control Signals

The control signals are a little harder to standardize, but some efforts have been made. There are basically two kinds of control signals, those that are synchronized to the processor and those that are not. The synchronized signals usually are associated with the video signal (e.g. start of line). The asynchronous signals are typically low rate (down to DC) signals that set the operating mode of the chip (e.g. the impulse response of the convolver).

Currently, the synchronous control signals include:

line start - goes high for one cycle at the start of each line

hsync - horizontal sync

hblank - horizontal blank

The line start signal is used to synchronize the delay lines and the contour tracer to the video line. It is also useful for applications requiring one operation per video line. An example of this is the incrementing of the refresh counter in the look-up-table RAM. The hsync and hblank signals are useful to control operations during the horizontal blank period of the video signal. Because there is no useful data during this period, it is often possible to suspend the signal processing and perform other operations. In the contour tracer, the blank time is used to write the entire line of data into the RAM. In the look-up-table RAM, the blank time is used to refresh the RAM. In both cases, the hblank signal is used to change the mode of operation (switch multiplexors, etc) and hsync is used as a write clock. All these signals should be latched into the chip during ph1.

So far, the asynchronous interfaces have been handled in two ways. The first way is to simply bring the control lines off-chip and allow the host computer to put a DC signal on these lines to determine the operation of the chip. The advantage of this scheme is that the chip can be operated without a host computer. That is, the operation can be set by jumper switches. This is also convenient during testing, when the host computer and the interface are no longer variables. To prevent the number of external control lines from growing too large, the number of degrees of freedom have to be limited. For example, a convolver was built with 16 preprogrammed impulse responses. In principle, for custom applications, simulations would be used to determine which impulse responses will be used and full programmability is not necessary and is even a hindrance.

Of course, there are people who claim that full programmability is essential. To achieve full programmability, a host computer interface was put on some chips. Because the host computer is operating asynchronously to the processors, an asynchronous interface is required. Further, the interface is required to be static as the control words must remain indefinitely. The standard interface has an 8 bit data word, a number of address bits that depends on the number of words stored on chip and a write select. The host computer puts the address of the internal register and the desired data on the bus and

then asserts the write select. An active low write select was chosen so that the interface could easily be implemented with a 74LS138. Usually, the data can be written into the internal registers at any rate below a maximum. Exceptions to this include the writing of the histogram equalization look-up table which must occur during the vertical blanking time to achieve real-time operation.

The timing for the programmable interface is shown in figure 11. This interface was used in the programmable convolver and the look-up-table RAM.

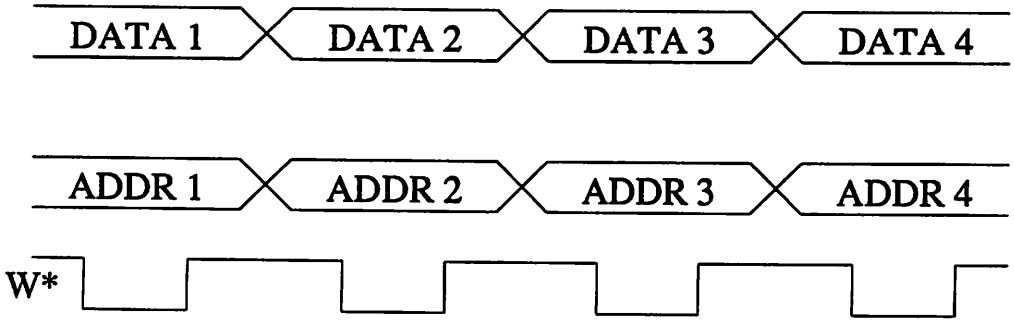



**Figure 11. Host Interface Timing**

### 7.2.3 Other Cases

The chips which do not input and output video signals have a more special purpose interface. The contour tracer and feature extractor fall into this category. The contour tracer operates on a video signal and hence its input interface is similar to that described above. The output and control interface is quite different and was designed for the tracing algorithm. The feature extractor was designed to interface between the contour tracer and a host computer. The host computer interface is similar to the asynchronous control interface described above.

# 7.3 A Single Chip 3x3 Convolver

## 7.3.1 INTRODUCTION

Many noise rejection, edge extraction and edge enhancement techniques are based upon linear FIR filters. In addition, 3x3 FIR filters have been used to find "interest" points in stereo images to determine the disparity between the two images [1]. A chip has been developed which performs a complete 3x3 convolution in real time on a 10 MHz video stream. A 3x3 convolver was chosen because it could be put on a single chip in 4μ NMOS technology. Also, many 3x3 masks have been examined and proven to be useful (see chapter 2). Further, longer impulse responses can be obtained by cascading several 3x3 convolvers. The convolver for 512 pixel lines occupies 47 $mm^2$ and dissipates 445 mW, while the convolver for 256 pixel lines occupies 35 $mm^2$ and dissipates 275 mW.

There are two ways of selecting the impulse responses. One chip has 16 impulse responses stored in a ROM. By using a ROM, the complexity associated with a computer interface for down-loading the coefficients is avoided. The 16 impulse responses initially put on the chip (see Listing 10, Appendix D) include 3 low-pass filters, 3 omni-directional high-pass filters, an edge enhancement filter, both Sobel operators, 3 all-pass filters for testing parts of the circuit and 4 compass gradient operators. A second chip has a fully programmable interface to demonstrate that there is no inherent reason why these filters can not be programmable. The programmable interface makes it possible to set each coefficient independently.

## 7.3.2 THE ARCHITECTURE

The convolver (see figure 1) consists of a gain stage, 2 line delays, 9 multiplying accumulators, 3 accumulators and controllers for the line delays and the arithmetic units. The 8 bit input data is first modified by the gain stage. The output of the gain stage is connected to the arithmetic units directly or

through the line delays. The data path is 10 bits wide inside the arithmetic units. The 8-bit result is then sent off the chip through a separate bus.

The gain stage scales the incoming data by a factor less than one ($1-2^{-n}$) to prevent overflow in the arithmetic units. By making overflow impossible, there is no need to include saturation logic in the arithmetic units. The gain stage also keeps the DC gain of the low pass filters near unity so that the average intensity of the image remains relatively constant. For most filters, the gain stage simply multiplies by unity.

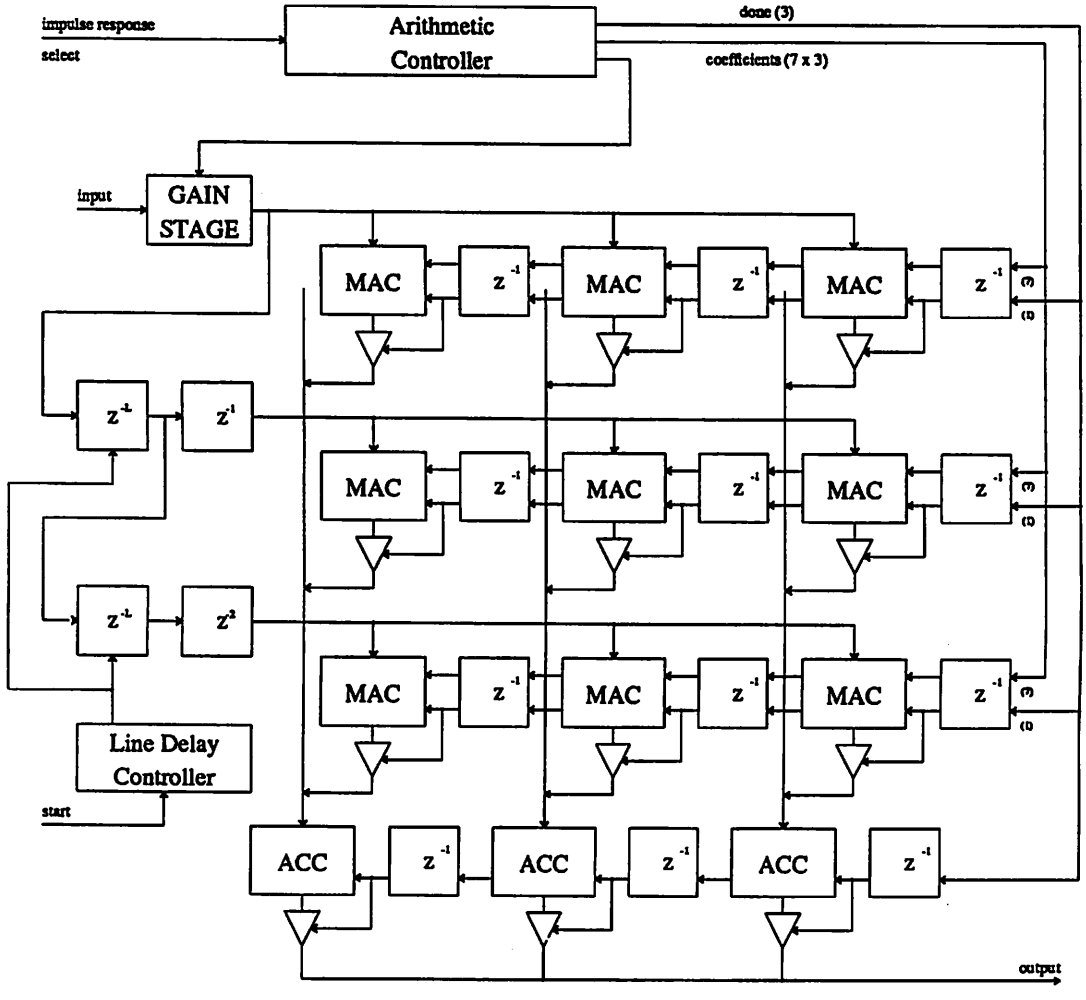



**Figure 1. 3x3 Convolver Architecture**

The line delays delay the video signal by exactly one line. The line delay controller generates all control signals necessary to shift in one line of data (512 pixels) while simultaneously shifting out one line of data each time the "start" signal occurs. The line delays are inactive after the proper number of

pixels have been shifted in/out until another "start" pulse is given. In this way, the blanking period of standard video can be accommodated. If "start" is held high, the convolver will free run for use with a sensor with no blank period.

There are two basic ways to organize the arithmetic units for operation at 10 MHz. The conceptually simplest way (see figure 2) would be to have all arithmetic units operating on the appropriate data for a single result. For example, this would mean that in a single cycle, the products $ah_{11}$, $bh_{12}$, $ch_{13}$, $gh_{21}$, $hh_{22}$, $ih_{23}$, $mh_{31}$, $nh_{32}$, $oh_{33}$ (see table 1) would be generated and accumulated for the result at pixel "h".

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - |
| - | a | b | c | d | e | f | - |
| - | g | h | i | j | k | l | - |
| - | m | n | o | p | q | r | - |
| - | - | - | - | - | - | - | - |

**Table 1a. Pixel labels for the example**

| | | |
|---|---|---|
| $h_{11}$ | $h_{12}$ | $h_{13}$ |
| $h_{21}$ | $h_{22}$ | $h_{23}$ |
| $h_{31}$ | $h_{32}$ | $h_{33}$ |

**Table 1b. Impulse response coefficients**

| | | |
|---|---|---|
| C | B | A |
| F | E | D |
| I | H | G |

**Table 1c. Processor Designations**

In the approach chosen, the multiplications being performed in a given cycle are actually for three different results. Instead of shifting the data, the coefficients are shifted and each arithmetic unit computes a different (offset by one or two pixels) 3x1 convolution over three cycles. That is, one unit (e.g. A) would compute $ah_{11}+bh_{12}+ch_{13}$ while the next (e.g. B) would finish computing $bh_{11}+ch_{12}+dh_{13}$ one cycle later and so on. Three units operating on different video lines (e.g. A,D,G) each compute one third of the entire 3x3 convolution. To get the final result, the outputs of the these three arithmetic units are summed and output. To decrease the number of additions needed to be performed by each accumulator in one cycle, the three units (A,D,G) output their results in consecutive cycles. This is accomplished by inserting an extra delay after the first line delay and two extra delays after the second line delay. These delay elements, along with a corresponding delay in control signals to the arithmetic units on each line,

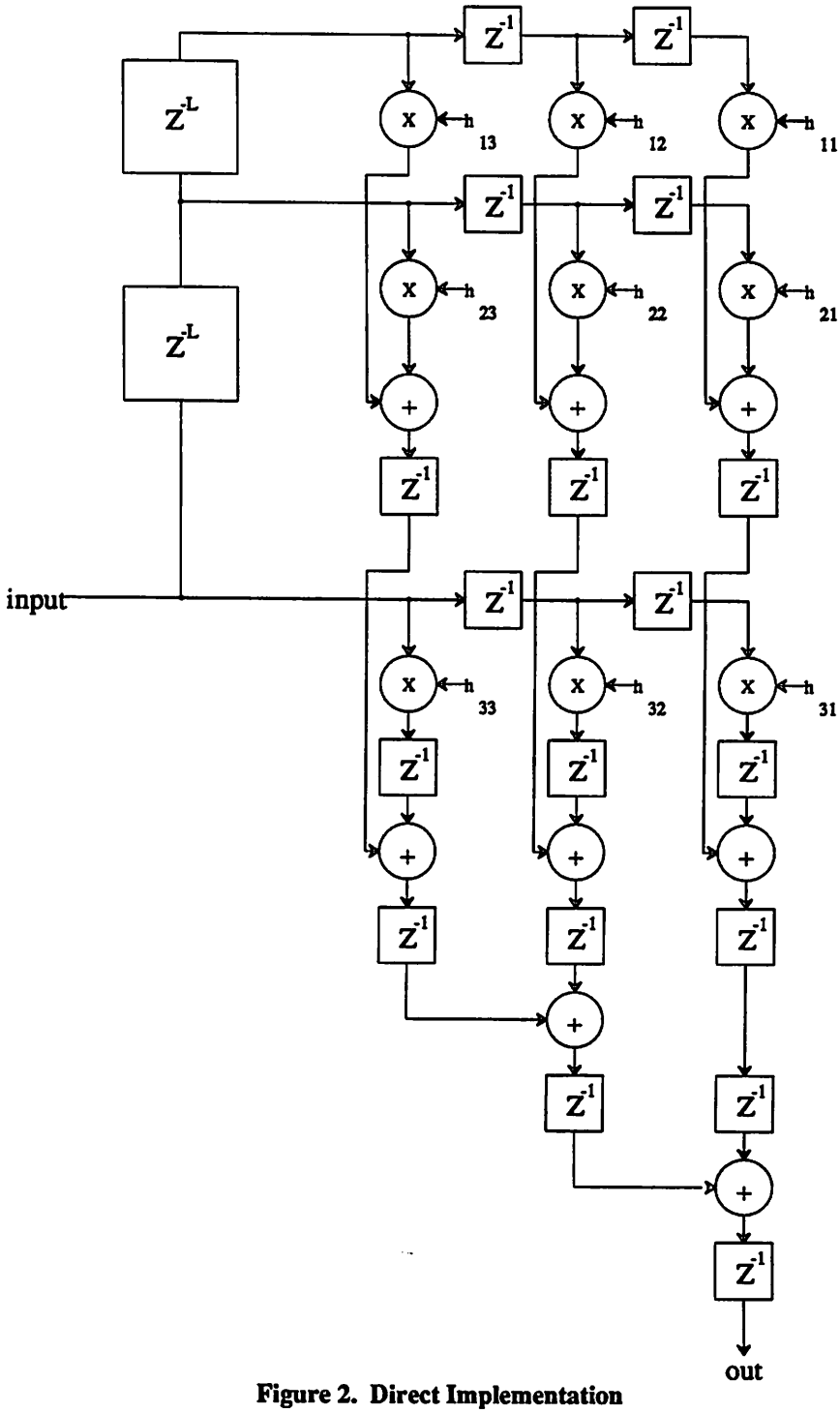



**Figure 2. Direct Implementation**

offset the results computed on each line. Instead of three results being valid on each output bus in a single cycle and none during the other two cycles, there is one result valid in each cycle. Finally, these three sums of 3x1 convolutions, computed by the accumulators, represent three complete results in three cycles. Therefore, real-time operation is achieved.

The arithmetic controller cyclically outputs the coefficients to the arithmetic units. It also outputs the "finished" signal which clears the accumulator and puts the result on the bus after a computation has been completed. Table 2a shows the coefficients at each arithmetic unit in the array for 5 consecutive cycles. Table 2b shows the data at each unit at the corresponding points in time. The value of the accumulator for the right-most arithmetic units is shown in table 2c and the accumulator values for the center row are shown in table 2d. It can be seen that the right row has a complete computation ready in cycle 6 while the center row has the next computation ready in cycle 7.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $h_{12}$ | $h_{13}$ | $h_{11}$ | $h_{13}$ | $h_{11}$ | $h_{12}$ | $h_{11}$ | $h_{12}$ | $h_{13}$ | $h_{12}$ | $h_{13}$ | $h_{11}$ | $h_{13}$ | $h_{11}$ | $h_{12}$ |
| $h_{21}$ | $h_{22}$ | $h_{23}$ | $h_{22}$ | $h_{23}$ | $h_{21}$ | $h_{23}$ | $h_{21}$ | $h_{22}$ | $h_{21}$ | $h_{22}$ | $h_{23}$ | $h_{22}$ | $h_{23}$ | $h_{21}$ |
| $h_{33}$ | $h_{31}$ | $h_{32}$ | $h_{31}$ | $h_{32}$ | $h_{33}$ | $h_{32}$ | $h_{33}$ | $h_{31}$ | $h_{33}$ | $h_{31}$ | $h_{32}$ | $h_{31}$ | $h_{32}$ | $h_{33}$ |
| 1 | | | 2 | | | 3 | | | 4 | | | 5 | | |

**Table 2a. Coefficients at each arithmetic unit over 5 cycles.**

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | b | b | b | c | c | c | d | d | d | e | e | e |
| x | x | x | g | g | g | h | h | h | i | i | i | j | j | j |
| x | x | x | x | x | x | m | m | m | n | n | n | o | o | o |
| 1 | | | 2 | | | 3 | | | 4 | | | 5 | | |

**Table 2b. Data at each arithmetic unit over 5 cycles**

| cycle | top (A) | middle (D) | bottom (G) | accumulator |
|---|---|---|---|---|
| 1 | $ah_{11}$ | x | x | x |
| 2 | $ah_{11}+bh_{12}$ | $gh_{21}$ | x | x |
| 3 | $ah_{11}+bh_{12}+ch_{13}$ | $gh_{21}+hh_{22}$ | $mh_{31}$ | x |
| 4 | $dh_{11}$ | $gh_{21}+hh_{22}+ih_{23}$ | $mh_{31}+nh_{32}$ | $ah_{11}+bh_{12}+ch_{13}$ |
| 5 | $dh_{11}+eh_{12}$ | $jh_{21}$ | $mh_{31}+nh_{32}+oh_{33}$ | $ah_{11}+bh_{12}+ch_{13}+$ $gh_{21}+hh_{22}+ih_{23}$ |
| 6 | $dh_{11}+eh_{12}+fh_{13}$ | $jh_{21}+kh_{22}$ | $ph_{31}$ | $ah_{11}+bh_{12}+ch_{13}$ $+gh_{21}+hh_{22}+ih_{23}$ $+mh_{31}+nh_{32}+oh_{33}$ |
| 7 | x | $jh_{21}+kh_{22}+lh_{23}$ | $ph_{31}+qh_{32}$ | $dh_{11}+eh_{12}+fh_{13}$ |

**Table 2c. Accumulator values for right row of array over 7 cycles**

This approach appears to be much more complicated than the direct approach, but it results in a very regular layout. Because the layout is very regular, the circuit can be easily assembled and modified for different impulse response lengths. On the negative side, 12 accumulators are required instead of 8 and the coefficients change at the data rate. In the first approach, the coefficients at each multiplier never change. However, to allow different impulse responses, the coefficients must be alterable at each multi-

| cycle | top (B) | middle (E) | bottom (H) | accumulator |
|---|---|---|---|---|
| 1 | x | x | x | x |
| 2 | $bh_{11}$ | x | x | x |
| 3 | $bh_{11}+ch_{12}$ | $hh_{21}$ | x | x |
| 4 | $bh_{11}+ch_{12}+dh_{13}$ | $hh_{21}+ih_{22}$ | $nh_{31}$ | x |
| 5 | $eh_{11}$ | $hh_{21}+ih_{22}+jh_{23}$ | $nh_{31}+oh_{32}$ | $bh_{11}+ch_{12}+dh_{13}$ |
| 6 | $eh_{11}+fh_{12}$ | $kh_{21}$ | $nh_{31}+oh_{32}+ph_{33}$ | $bh_{11}+ch_{12}+dh_{13}$ $+hh_{21}+ih_{22}+jh_{23}$ |
| 7 | x | $kh_{21}+lh_{22}$ | $qh_{31}$ | $bh_{11}+ch_{12}+dh_{13}$ $+hh_{21}+ih_{22}+jh_{23}$ $+nh_{31}+oh_{32}+ph_{33}$ |

**Table 2d. Accumulator values for center row of array over 7 cycles**

plier and the second approach provides a simple means of achieving this.

To see the importance of a symmetric layout, one need only look at the number of busses connecting to the 9 multipliers. There are 3 input data busses of 8 bits each, 9 coefficients of length depending on the encoding scheme and 9 outputs of 10 bits each. If not done properly, these connections could be an enormous mess.

One way of looking at these two approaches is that the direct approach has a single word micro-program while the chosen approach has a three word micro-program.

### 7.3.3 ARITHMETIC UNIT

Each arithmetic unit (figure 3) consists of a multiplier, an accumulator and pipeline registers. Because the coefficients are limited to powers of two, the multiplier is very small (15 transistors/slice) and consists of a barrel shifter (to control the magnitude of the coefficient) an XNOR gate (to control the sign of the coefficient) and a pull-down transistor to allow zero coefficients. The adder is a ripple-carry type with different even and odd bit slices. The output bus is pre-charged high and can only be pulled-down by the selected arithmetic unit. There are 7 control lines to specify the coefficient and a single control line to reset the accumulator and put the result on the output bus. The arithmetic units that perform the accumulations (bottom of figure 1) are similar to those of figure 3 without the multiplier section.

It would be possible to allow greater precision in the coefficients without going to a full parallel multiplier. Another multiplier-adder section could be added to the arithmetic unit for each additional "1" allowed in the coefficient.

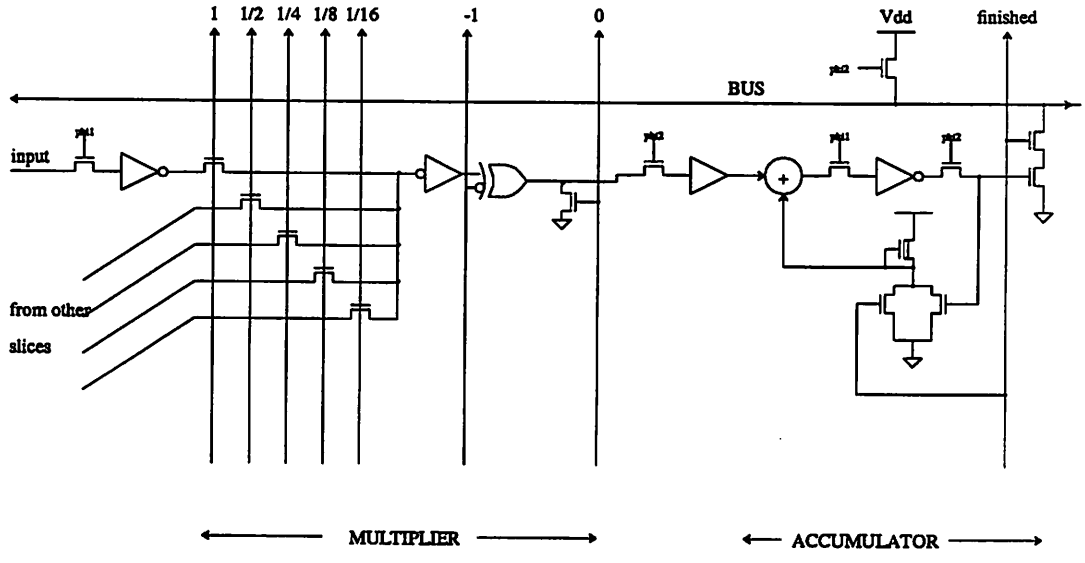



**Figure 3. Arithmetic Unit**

### 7.3.4 ARITHMETIC CONTROLLER

The ROM-based arithmetic unit controller (figure 4) consists of a 3-bit shift register that provides the micro-program address and a NOR-NOR ROM that contains the coefficient data and other control signals for the 16 3x3 impulse responses. A shift register was chosen over a counter and decoder because the shift registers accomplishes both functions in less space than the counters alone would occupy. By choosing the impulse response number to be the row address and the micro-program address to be the column address, a very fast low-power circuit could be designed. As the impulse responses changes at a very low rate, the row selects and bits lines have a long time to settle. The column decoder and output register are the only parts that must settle quickly. By keeping the impulse response select separate from the high speed part of the controller, the number of impulse responses stored on chip can be increased without performance degradation.

The programmable arithmetic controller consists of a 3-bit shift register for selecting the current micro-code word and an array of static registers, 10 words of 8 bits each, to hold the micro-code for a single filter. There is one register for each coefficient (7 bits for the coefficient and 1 bit for the "finished" signal) and a single register to set the gain stage operation. Fully static selection logic on chip makes the registers appear like a write only static RAM. Because the programmable interface does not need to be written at image data rates, low-power circuits could be used for the selection logic and for the latches

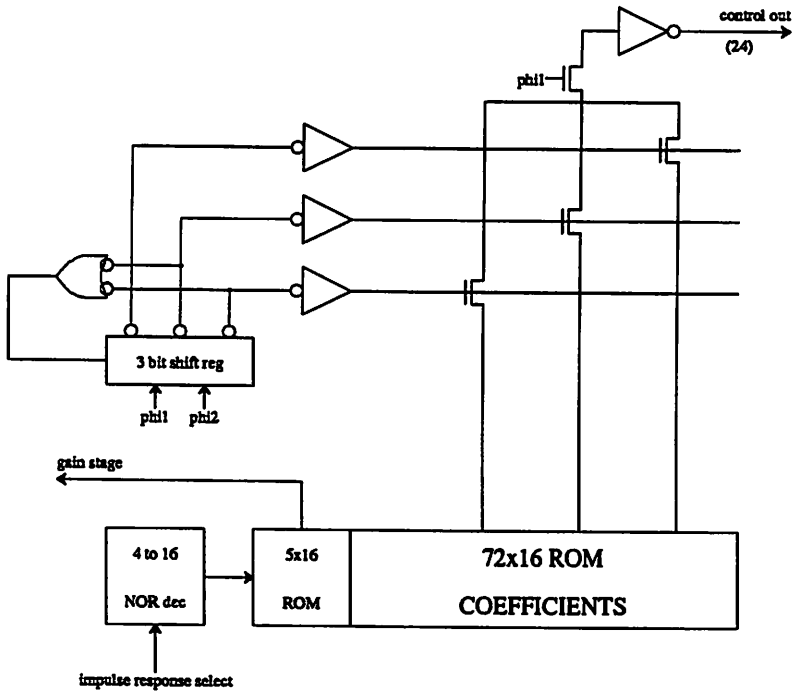



**Figure 4. ROM based Arithmetic Controller**

(NAND type cross-coupled circuits were chosen).

### 7.3.5 EXPANDING BEYOND 3x3

One question that is often asked concerns the possibility of expanding the convolver beyond the basic 3x3 configuration. There are three basic approached that can used. First, for small expansions that require more operations per line but not over more lines, it is possible to put more MACs on the original 3x3 convolver chip. A 3x5 kernel would fit on a single chip in 4μ NMOS technology. The second approach would be to break the problem up in a different way (figure 5). Instead of trying to make each chip a self contained functional block, the line delays could be put on one chip and the arithmetic units on another. The line delay blocks could be cascaded to operate over as many lines as desired. Further, for operators that operate on the same input data (e.g. both Sobel operators), the line delays would not be duplicated as they are when using the basic 3x3 convolver chips. The third possibility would be to keep the basic 3x3 chip, but bring out the line delay outputs so that several convolvers can operate over different lines (figure 6). Another input could be used to sum up the outputs of the different convolvers to achieve the complete output. This scheme has the advantages that all the blocks (3x3 convolvers) are the same and are also functionally complete as a single chip. The line delays may be repeated, however,

leading to a somewhat less efficient use of the silicon. A final approach is to simply cascade existing 3x3 chips. In this way, a $n \times n$ operator requires only $\frac{(n-1)}{2}$ chips. However, an arbitrary impulse responses can not be obtained for n>3 as there are only $\frac{(n-1)}{2} * 9$ parameters to determine $n^2$ coefficients (all of which currently must be powers of 2).

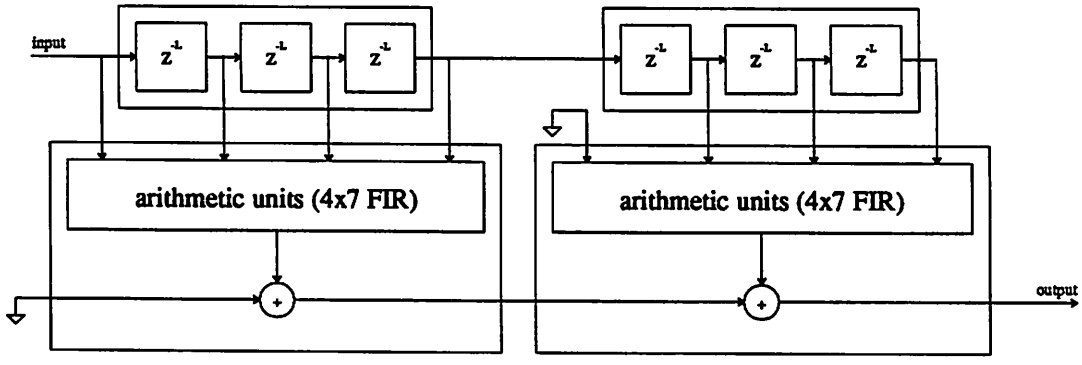



**Figure 5. Expanding to 7x7 with 4 Chips and 6 Line Delays**

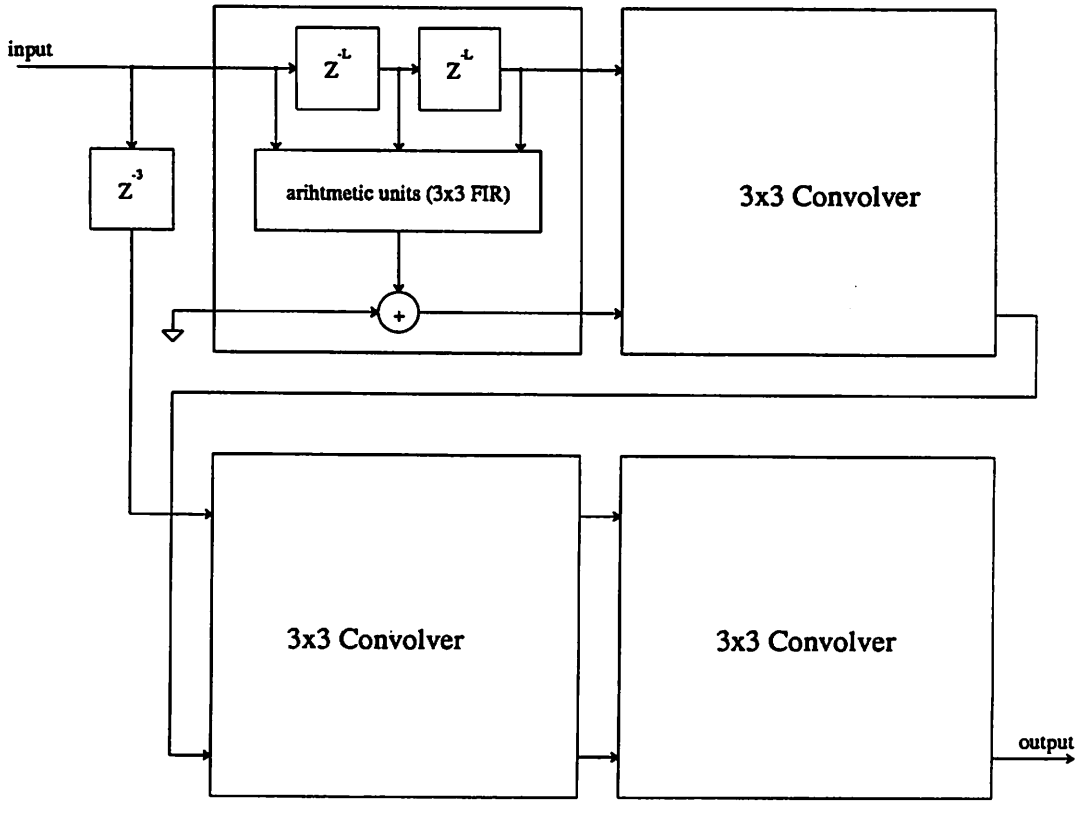



**Figure 6. Expanding to 5x6 with 4 Chips and 8 Line Delays**

### 7.3.6 POST PROCESSORS

A post processor was developed to provide a complete Sobel operator using 2 convolvers, more general edge enhancement, edge extraction and output format control.

The chip (figure 7) has two signal inputs and a single signal output. Both input signals can be inverted or zeroed or the absolute value can be found. In addition one signal can be multiplied by a power of two between 1 and 8. After this processing, both signals are summed. Full control of these functions is not possible in current chips due to pin limitations. If a host processor interface is put on chip, all functions could become available.

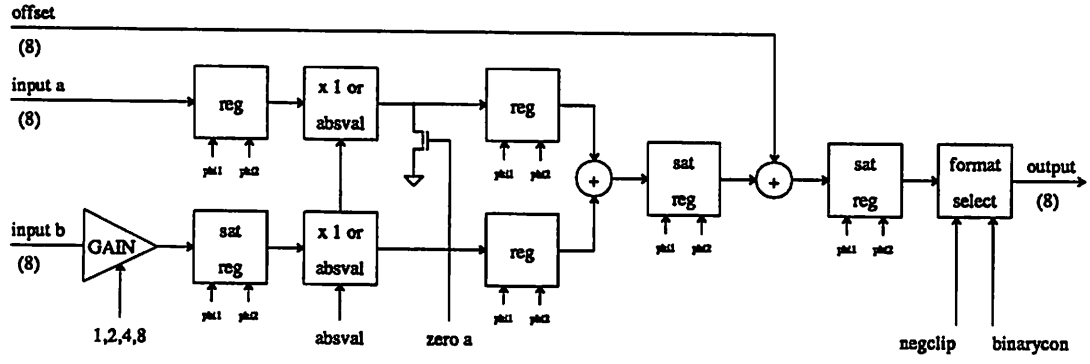



**Figure 7. Sobel Post Processor Block Diagram**

After being summed, a constant is added to the signal and one of three output transformations can be chosen. The constant makes is possible to add offsets to the signal or to adjust the threshold depending on the output format chosen. The first output format, called negative clip, displays positive grey levels over the entire range of output grey levels available. This the normal mode of operation. The second output format displays the negative grey levels over the lower half of the output grey level range and the positive grey levels over the upper half. This allows the outputs of filters that generate negative values (e.g. high-pass filters) to be displayed. The third (called binary clip) zeros out all lower bits to yield a binary or thresholded image.

To perform a Sobel operator, the output of the Sobel-X and Sobel-Y operators are passed through a full wave rectifier and summed. Edge enhancement can be performed by adding a omni-directional high-pass filtered version of the image to the original. The degree of edge enhancement can be varied by varying the gain in the high-pass filter path. To extract edges, the output of a high-pass operator is simply

thresholded. The threshold can be varied from off the chip.

The active area of this chip is 1.5 $mm^2$ and and power consumption is 200 mW (including pads).

### 7.3.6.1 Non-Linear LPF Post Processor

A second post processor (figure 8) was designed to perform non-linear low-pass filtering (see section 2.5.2) from a low-pass and an all-pass input. The technique is to compare the current pixel with the average of its eight neighbors. If the difference is greater than a threshold, the current pixel is considered to be "bad" and is replaced by the neighborhood average. Otherwise, the current pixel is left alone.

The original convolvers did not have a filter that computed the average of eight neighbors (all LPF included the current pixel). However, a Laplacian filter is actually the difference between the desired low-pass and the all-pass filters. By subtracting the Laplacian filter output from the all-pass signal, the neighborhood average is obtained.

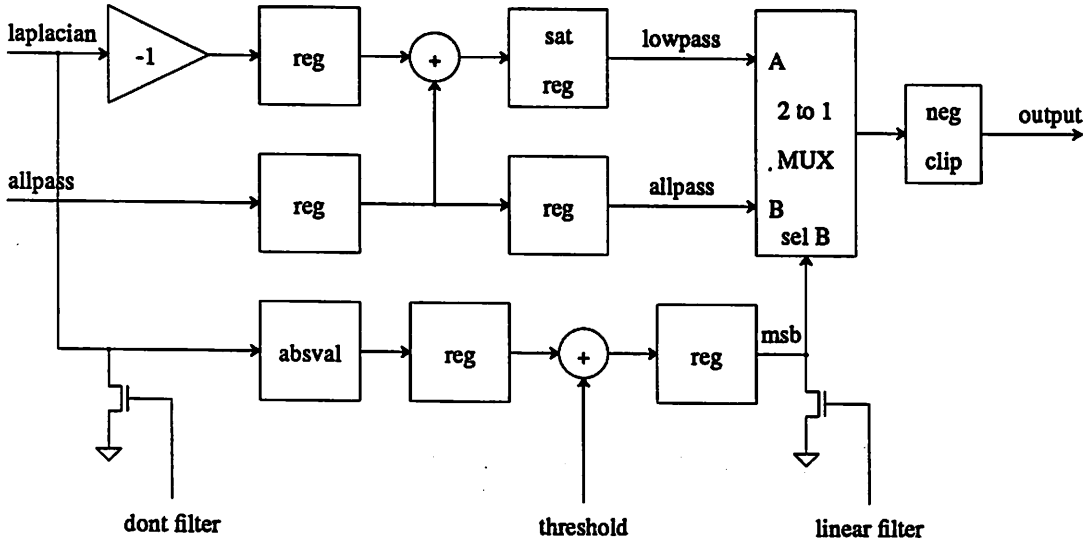



**Figure 8. Post Processor for Non-Linear Low-Pass Filter**

The processor determines if the current pixel is "bad" by thresholding the absolute value of the Laplacian filter output. This decision controls a multiplexor that either passes the all-pass signal or the difference of the all-pass and Laplacian signals (low-pass signal).

This circuit was generated by a data-path compiler that worked from a description of the desired operations. It then placed all blocks and performed the routing, making it possible to create the chip layout in a matter of hours.

The active area of this chip is 2 $mm^2$ and power consumption is 150 mW (including pads).

### 7.3.7 REFERENCES

[1] Bandyopadhyay, A., "Interest Points, Disparities and Correspondence," *Proc. Image Understanding Workshop*, Oct. 1984, pp. 184-187.

# 7.4 A 7x7 Logical Convolver

## 7.4.1 INTRODUCTION

The logical convolver performs two-dimensional logical operations on binary images in real-time (10 Mhz for 512x512 images). The operator has a mask of size up to 7x7. Currently, the chip performs nearly circular "dilation" or "bloating" type operations but could be easily reconfigured for other types of operations. Dilation is used to join broken edges after edge extraction and to reduce the bandwidth of the edge map before down sampling. The logical convolver occupies 18 $mm^2$ and consumes 130 mW.

Logical convolution has an affect similar to a two-dimensional FIR low-pass filter that operates on binary images. That is, a point in the input image is spread into a finite pattern specified by the binary mask or impulse response. The filter is, however, non-linear so that the output is not the sum of responses to the individual points in the input image. Rather, the output is the "OR" of the responses to the individual points.

Logical convolution is simply a degenerate case of linear convolution in which the input data, output data and impulse response are all single bit signals. To perform two-dimensional FIR filtering, the input values are multiplied by the corresponding value of the impulse responses and summed. If the input values and impulse response values are both single bit, the multiply operation is identically an AND operation. The sum of these single bit multiplies is, however, multi-bit. To get the desired number of bits at the output, simple saturation is used. That is, if the sum is greater than or equal to 1, it is set equal to 1. Otherwise, it is set equal to 0. This is identically an OR operation. Therefore, the logical convolver is just a linear convolver with the multipliers replaced by AND gates and the adders replaced by OR gates.

The basic difference equations for dilation and erosion are given below. The equation for dilation

is the same as that for linear convolution with the sum replaced by an OR operation and the multiplication replaced by an AND operation.

$$\text{dilation} \quad g_{x,y} = \underset{n,m}{OR}(h_{D\,n,m}\ AND\ f_{x-n,y-m})$$

$$\text{erosion} \quad g_{x,y} = \underset{n,m}{AND}(h_{E\,n,m}\ OR\ f_{x-n,y-m})$$

The equation for erosion can be transformed into:

$$\text{erosion} \quad \overline{g_{x,y}} = \underset{n,m}{OR}(\overline{h_{E\,n,m}}\ AND\ \overline{f_{x-n,y-m}})$$

If the input, output and mask values are inverted, the erosion equation is identical to the dilation equation and both operations can be performed with a single logical unit (with the addition of an XOR gate at the input and output). Essentially, erosion is the expansion of the background. Inverting the background and foreground and expanding the foreground and then inverting the background and foreground again is equivalent to expanding the background.

### 7.4.2 WHY A CUSTOM ARCHITECTURE IS NEEDED

To perform a 7x7 logical convolution in real-time, the equivalent of 49 memory reads, 49 logical ANDs, 48 logical ORs and 1 memory write must be performed per sample. At 10 MHz this adds up to about 1.5 Gops/Sec. It is unlikely that such a large rate of logical and memory operations could be performed on a general purpose processor. The linear convolver could be used to perform a logical convolution over a 3x3 region if the output is clipped to the same magnitude as the input. This should be clear because logical convolution is a case of linear convolution. However, the linear convolver has multi-bit line delays and data-paths and has multipliers which can multiply by $+/-2^{+/-n}$. All of this extra hardware ( $540\frac{transistors}{mask\ elements}$ ) will be unused when logical convolutions are performed. To keep the size of the processor as small as possible so that the logical convolution can be performed over a larger region, a custom architecture is used which requires only $10\frac{transistors}{mask\ elements}$. By using single bit storage elements and logical processors a full 7x7 logical convolver requires only $\frac{1}{4}$ the area of the 3x3 linear convolver.

Although the logical convolver is a natural extension of the linear convolver, there is virtually no similarity in the actual circuit. In the 3x3 linear convolver, the data was held in place while the coefficients were shifted. This resulted in greater uniformity of layout because the outputs of several units could be utilized in a regular manner. In this case, the outputs of the individual AND gates can be combined in a straight forward manner using a direct implementation. Because the data is a single bit wide, the problems associated with routing are greatly reduced.

### 7.4.3 HARDWARE

The core of this circuit is the line delay originally designed for use in the linear convolver. Although it was designed as a 1-line x 8-bit delay, by wiring the first bit output to the second bit input, and so on, it could be changed into an 8-line x 1-bit delay element (figure 1). The output of each of the 7 delay lines is input to a set of shift registers that are 7 bits long (figure 2). At the output of these 7 shift registers are the 49 points that are used to compute the current output by the logic array. Each of these 49 points is multiplied by the corresponding mask or impulse response value. The outputs of all multipliers are then ORed together to get the output. Currently, the logic array is set up for a circular expand by 0,1,2 or 3 that is chosen by the user. However, this could be altered for any type of impulse response either mask or user programmable.

The logic array is very compact (figure 3). Each shift register output is gated with the appropriate mask value. All gated shift register outputs on the same video line are wired ORed together. Because the OR gate is fairly long a pre-charge device pulls up the output node quickly. After the OR gate a pipeline register is used to reduce the critical path length. The outputs for each line are then wired ORed together to produce the final result.

### 7.4.4 HARDWARE PARAMETERS

Parameters which can vary include the operator size (N), the line length (L) and the degree of freedom allow in the impulse responses. The affect of L on the hardware has been discussed already in relation to the delay line. N affects the hardware in two primary ways. First, a different number of line delays is required. If N<7, the problem of changing the standard line delay for fewer bits arises. For

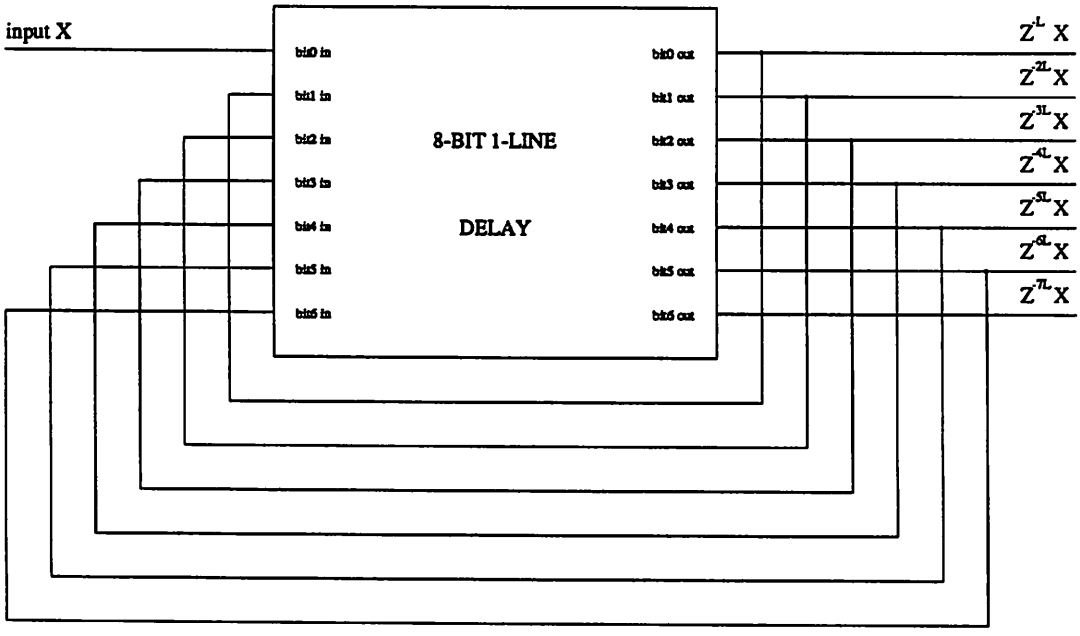



**Figure 1. Line Delay Organization**

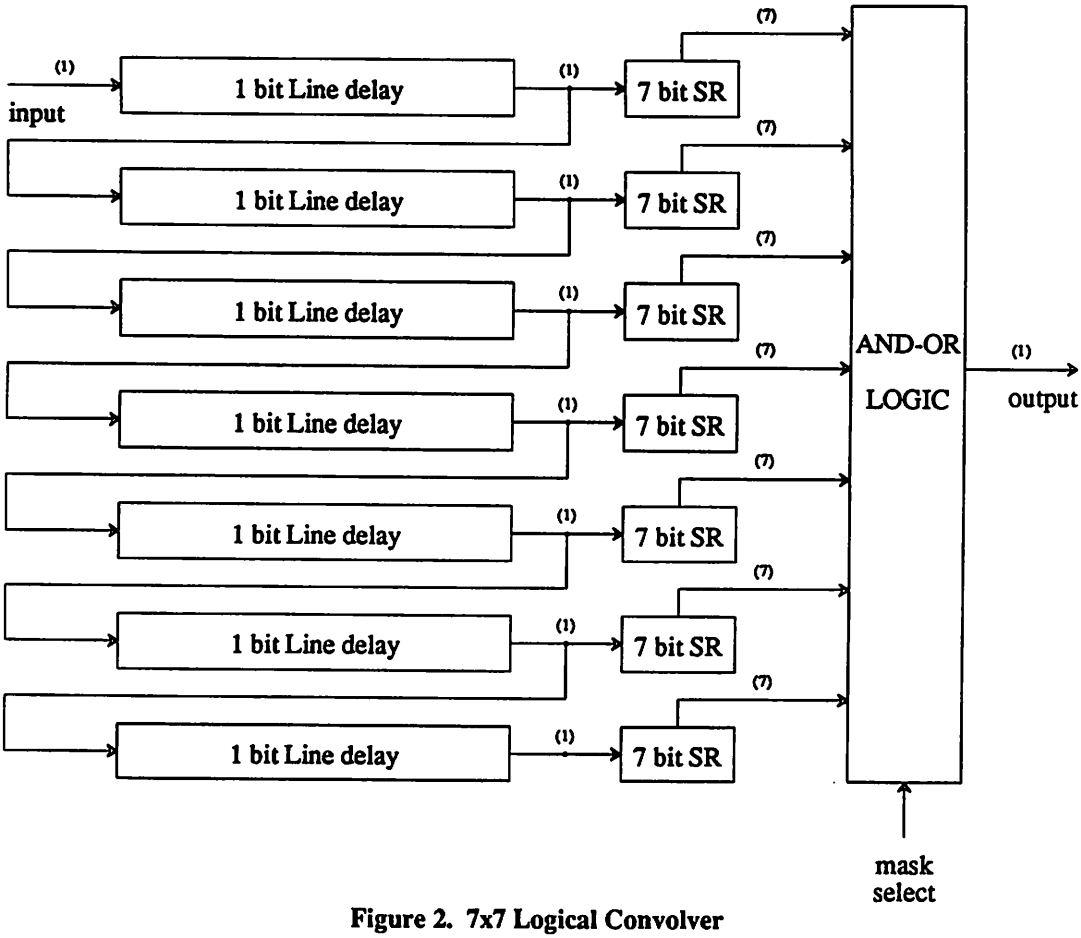



**Figure 2. 7x7 Logical Convolver**

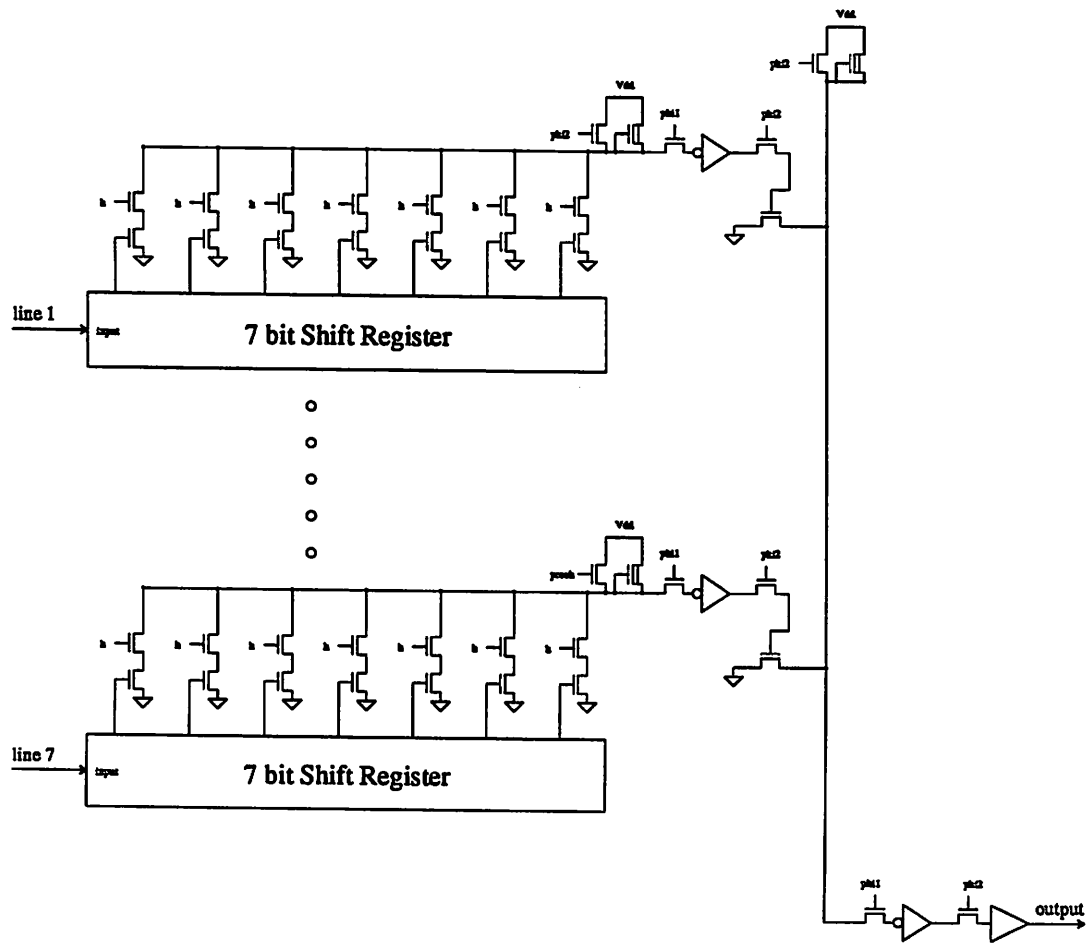



**Figure 3. Logic Array**

N>7, either more bits can be added to a single line delay or more line delays can be added. N also affects the number and size of the shift registers.

The degree of freedom given to the impulse response can make a large difference. At one extreme, a single impulse response can be hard-wired by including connections or not in the logic array (hard-wired AND gates). The next step, is to allow some parameterizable class of impulse responses, such as circular expand or shrink. In this case, transistors are included in the logic array to perform the AND operations, but many fewer than $N^2$ parameters are needed to describe the impulse response. For circular expand and shrink operations only 4 parameters are required. This means that only 4 wires had to routed to the logic units. In the fully programmable mode, $N^2$ wires will be routed to the logic arrays and all $N^2$ AND gates would be programmable. Some form of host interface would be required on chip since 49 pins can not be used to define the impulse response. For a 7x7 operator, full programmability could be a

significant cost that should be justified with corresponding gains in performance. In the fully programmable case with large N, the architecture of the linear convolver may be preferable since only N wires would be needed to connect the host interface and the logic array. However, the logic unit and the interface would both increase in complexity.

### 7.4.5 PERFORMING OTHER LOGICAL OPERATIONS

The expanding and eroding operations are a subset of neighborhood operations performed on binary images. A more general approach would be to take all the pixels that enter into the computation of the current output and form a binary word from the string of binary values. This word can then be used as an address to a ROM with a single output, the result. The ROM could then be programmed for any function of the neighborhood pixel values. This approach is reasonable for a 3x3 neighborhood as the ROM would be 512 bits. However, for larger neighborhoods, the ROM size becomes prohibitive. The 7x7 processing would require a ROM with an area of 112000 square meters.

Shown in table 1 are the masks for expanding by 0,1,2 or 3. If the input is a single dot, the output of the logical convolver will be the same as the mask.

| mask0 | mask1 | mask2 | mask3 |
|---|---|---|---|
| 0000000 | 0000000 | 0000000 | 0001000 |
| 0000000 | 0000000 | 0001000 | 0011100 |
| 0000000 | 0001000 | 0011100 | 0111110 |
| 0001000 | 0011100 | 0111110 | 1111111 |
| 0000000 | 0001000 | 0011100 | 0111110 |
| 0000000 | 0000000 | 0001000 | 0011100 |
| 0000000 | 0000000 | 0000000 | 0001000 |

**Table 1. MASKS**

| mask0 | mask1 | mask2 | mask3 |
|---|---|---|---|
| | | * | * |
| | | *** | *** |
| | * | ***** | ***** |
| * | *** | ******* | ******* |
| * | *** | ***** | ******* |
| | * | *** | ***** |
| | | * | *** |
| | | | * |

**Table 2. Output for input image with 2 dots**

# 7.5 A 3x3 Non-Linear Filter Based on Sorting

## 7.5.1 INTRODUCTION

Some image processing techniques are based on sorting the elements in some neighborhood of pixel and taking the Nth largest of the sorted values as the new value for that pixel. For noise with very long tails in its distribution, median filtering has been shown to be very useful [1]. Median filters possess the desirable property that monotonic changes in the signal are preserved but sharp spikes are removed. Therefore it is possible to filter the image to remove noise without blurring the edges and making edge extraction more difficult. Median filters have been shown to be inferior (in the MSE sense) to an averaging filter [1] for some cases when the noise has a Gaussian distribution. The use of the minimum and maximum over a region has been proposed for noise rejection filters [2]. The difference between the maximum and minimum (maximum difference) can be used for edge extraction and other high-pass processing.

## 7.5.2 HARDWARE REQUIREMENTS

To completely sort the 9 pixel values in the 3x3 window requires at least 29 comparators [1]. The 29 comparators must be wired in a fairly complex manner, resulting in a large cost. Other schemes [3] use more comparators (36) in a bubble sorting network, but achieve simpler wiring. However, if not all of the 9 sorted values are needed, fewer comparators can be used. For example, the minimum or maximum of a 3x3 region can be computed with only 8 comparators. The number of comparators can be reduced further by noting that the entire maximum does not need to be recomputed for each pixel. Adjacent pixels have six pixels in common that go into the maximum or minimum computation. The maximum, or minimum, of these six pixels can be re-used to save comparators.

One way to reduce the number of comparators is to make the filter separable. That means that a cascade of two filters is used. The first operates in one dimension while the second filter operates in the other dimension. The net result is that only 2 one-dimensional filters are needed instead of one two-dimensional filter. The reason that fewer processors are needed for the separable filter is that each output of the first filter is used n times by the second filter instead of being recomputed. The non-separable filter requires $n^2$-1 processors to compute the maximum or minimum over a $nxn$ region while the separable filter requires only $2(n-1)$ processors. Because the results of the first filter of the separable filter are used n times by the second filter, the effective number of processors in the separable filter is $(n-1)n+(n-1)$ or $n^2-1$, which is the same as the number of processors in the non-separable filter.

Making the maximum, or minimum, processor separable does not alter the results of the computations. The maximum of one-dimensional maximums (i.e. the separable maximum) is the true 2 dimensional maximum; this is also true for the minimum. Although the separable median is not the true 2 dimensional median, it will still reject extreme points while preserving edges. To compute the 3 pixel one-dimensional maximum, median, and minimum (i.e. a three element sort) requires only 3 comparators. Therefore, the separable maximum, minimum, and median can be computed with only 6 comparators that can be easily wired. If the median is not required, only 4 comparators are needed to compute the maximum or minimum over 3 pixels.

### 7.5.3 CHOSEN ARCHITECTURE

The chip that was developed has a total of 10 comparators: six compute the separable maximum or median, and the remaining four simultaneously compute the separable minimum. Therefore, the maximum difference can be computed. The basic processing element used is the two-way sorter (figure 1). It takes in two values and outputs the minimum and maximum of the values. Three of these can be connected to form the three-way sorter (figure 2). Two three-way sorters can then be connected to give the complete separable filter (figure 3). One filter processes 3 pixels from the same vertical line. That is, the inputs to the filter are the current pixel and the pixels delayed by one and two lines. The second filter processes three pixels from the same horizontal line. Its inputs are the current pixel and the pixels delayed by one and two pixels.

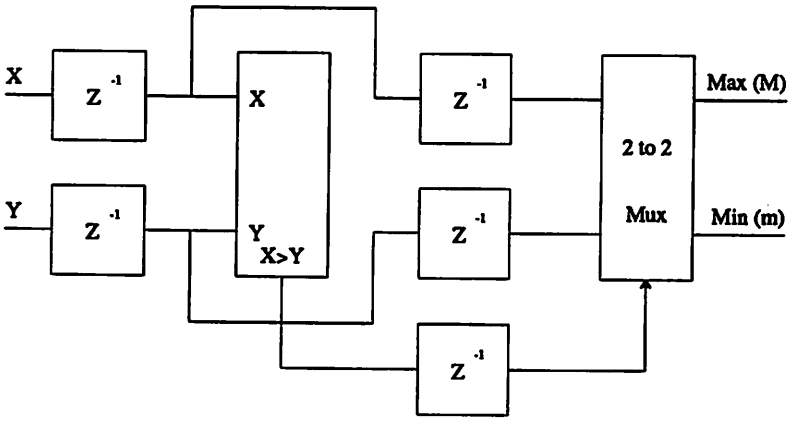



**Figure 1. 2-Way Sorting Element**

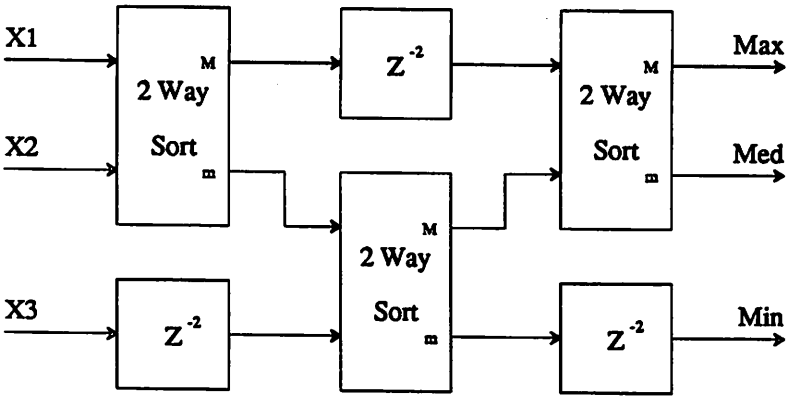



**Figure 2. 3-Way Sorting Element**

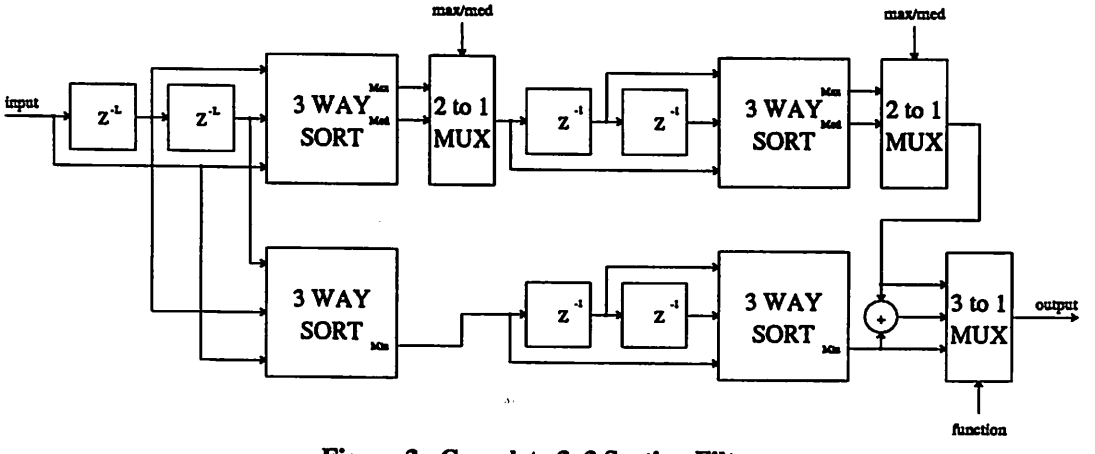



**Figure 3. Complete 3x3 Sorting Filter**

### 7.5.4 REGION OF OPERATION

The user has limited control over the mask of the filter (see table 1). Essentially, the mask indicates which pixels will be included in the computations. Only a single bit is needed to represent each

element of the mask because it indicates only whether the pixel will be included or not in the current output of the filter. Because the filters are separable, the user can only choose which points in each of the 2 3x1 regions are operated upon. Of the 64 possible masks, only 16 are really useful because many of the masks are equivalent and some mask out all points from the 3x3 region and produce a constant zero output. Therefore, only 2 bits are used to represent the mask of each of the 1D filters. Essentially, one mask bit in each of the two filters is set so that the corresponding pixel is never masked out. The net result of this masking scheme is that the filters operate only over rectangular subregions of the 3x3 region.

The center pixel in vertical processor was not chosen to be always included in the computations for testing purposes. By always including data from the current line and being able to ignore data from each of the line delays it is easier to test the line delays and arithmetic units separately. For example, if the first line delay is defective, the data from both will be corrupted. In this case the only way to test the arithmetic units is to disable both line delays.

The mask is implemented with AND gates that mask out values from the maximum computation (adding zeros does not change the maximum) and OR gates that mask out values from the minimum computation (adding full scale values does not affect the minimum). There is no comparable way of masking values for the median calculation so the median must be computed over the entire 3x3 region.

Because the filter masks are implemented by inserting full scale or zero values, the vertical minimum must be computed in a separate processor. The zeros inserted to mask out values from the maximum computation would force the minimum to always be zero. If the sorter always operated over the entire 3x3 region only a single vertical processor would be needed and 2 comparators would be saved.

### 7.5.5 COMPARISON TO THE LINEAR CONVOLVER

Although this sorting circuit and the linear convolver both perform some operations on the pixels in a 3x3 region, the architecture of the two circuits is quite different. First, the linear convolver is micro-coded and requires a controller that operates at the sample rate while the sorter operates in a fully parallel mode with no controller. The micro-coded approach was chosen for the linear convolver to minimize the

complexity of signal routing and coefficient routing. This was a major concern because each of the 9 arithmetic units requires a multi-bit coefficient and the outputs of all nine arithmetic units must be summed to obtain the result. The architecture chosen made the routing very regular and hence easy to perform. The sorter has only four single bit coefficients and the outputs of the individual processors connect only to nearby processors. Further, the median is more difficult than the maximum or minimum to compute over several cycles by a single micro-coded processor. The maximum and minimum can be computed from only the current input and the partial maximum or minimum. The median can not be computed in a similar manner and must have more data stored. With these differences, the parallel implementation proved simpler.

### 7.5.6 CHIP GENERATION

This circuit could be created quite quickly, as the line delays and line delay controller were the same as those used in both the linear and logical convolvers. The arithmetic units (sorters) were assembled with a data-path generator using some of the same basic cells as those used in the linear convolver. To reduce the size of the sorters, special cells were generated. Double registers, buffers and multiplexors (double cells have circuitry for 2 signals in a single bit slice) were created because the comparators and switching circuits operate on two signals at time in a symmetric way. All routing wires for the two element sorter are included inside the double cells and no global routing was needed. Further, the double cells are less than twice as large as the single cells because every cell requires a certain amount of overhead regardless of the number of transistors it contains. The net result of using the double cells is that space is saved by both the more efficient use of the cell space and the reduced need for global routing.

The separable filter architecture made the bit-sliced data path approach more feasible. The data-path compiler requires that all blocks in a bit slice be physically arranged in a line. This can always be done for an arbitrarily complex network, but may require complicated and area inefficient routing. Because the separable sorting filter has blocks that are connected to near neighbors, the routing can be done with only a few tracks. Non-separable filters have more global routing and hence could not be as efficiently implemented with the data-path generator.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

**Table 1. Masks or regions of sorter operation**

### 7.5.7 REFERENCES


[1] P. M. Narendra, "A Separable Median Filter for Image Noise Smoothing", *Digital Image Processing* and *Analysis*: *Volume* 1: *Digital Image Processing*, pp. 450-459, 1985.

[2] Healy G., Sanz J. L. C., "CONTAM: An Edge-Based Approach to Segmenting Images with Irregular Objects," *IBM Research Report*, Jan. 1985.

[3] N. Demassieux, et al, "VLSI Architecture for a One Chip Video Median Filter", *Proc. of ICASSP*, pp. 26.7.1-26.7.4, 1985.

# 7.6 An Image Contour Tracing Chip

## 7.6.1 INTRODUCTION

To recognize two-dimensional images that are characterized by their closed contours, the curvature of the contour is useful as a feature. The curvature is a one dimensional signal that can be matched using techniques from speech processing to provide insensitivity to translation, rotation and size. To generate the curvature from the video image, a contour tracing chip has been developed.

The discrete curvature was defined in section 3.2 and is given below:

$$curv(n) = \frac{\Delta\phi(n)}{\Delta n} \text{ where } \phi(n) = \frac{1}{\frac{\pi}{4}} \tan^{-1} \frac{\Delta Y(n)}{\Delta X(n)}$$

Because $\Delta X$ and $\Delta Y$ can only be -1, 0 or +1, the curvature is an integer between 0 and 7 (or -4 and 3). By representing the contour by its curvature, not only does one get a signal that has useful properties for image recognition but also a representation of the contour which requires many fewer bits than a direct representation based upon the X and Y coordinates. If the initial conditions $X(0)$,$Y(0)$ and $\phi(0)$ are known, the contour can be reconstructed from the curvature in the following way:

$$\phi(n) = \phi(0) + \sum_{i=0}^{n} curv(i)$$

$$X(n) = X(0) + \sum_{i=0}^{n} \Delta X(i) \quad \text{where } \Delta X(i) = \sin\left[\frac{\pi}{4}\phi(i)\right] \Delta s(\phi(i))$$

$$Y(n) = Y(0) + \sum_{i=0}^{n} \Delta Y(i) \quad \text{where } \Delta Y(i) = \cos\left[\frac{\pi}{4}\phi(i)\right] \Delta s(\phi(i))$$

$\Delta s(n)$ is the length of the contour between two the points at position $n$ and has a value of either 1 or $\sqrt{2}$. Because both $\phi$ and *curv* are 3 bit numbers, $\Delta X$ and $\Delta Y$ can be easily computed with a 8 word look up table.

### 7.6.2 THE ALGORITHM

The contour tracing algorithm which was implemented is quite simple. First, the edge map image is loaded into the internal RAM after being down sampled to 128 x 120. Then, starting from a user preset location, the controller starts raster scanning in search of an object in the image. When an object is found (i.e. a non zero pixel is found), pixels in the neighborhood of the current pixel are checked. The pixels are checked in an order which guarantees that the first pixel found to be "on" will be on the outside of the contour [1]. This "on" pixel is then made the current pixel and the process repeats until the contour has been traced.

The contour is traced in a generally counter-clockwise direction. The search for the next pixel proceeds in a counter-clockwise direction around the current pixel and starts at the pixel in the most clockwise position that was not checked in the previous search. In this way, the tracer always stays on the outside of the contour (some schemes [2] do not do this) and can not get stuck in holes in the interior of the object.

Figure 1 shows the definition of direction vectors and the steps that the tracer will make to find the next pixel on the contour as a function of the direction between the last two points on the contour. The points marked with "*" indicate the last two points found on the contour. The center point was the last point found. The numbered points indicate the order that the points will be tested. The "x" points were tested in the previous cycle and do not need to be tested again. An exception is for direction 0 when the contour is first encountered after raster scanning and the "x" points are checked. It can be seen that the X and Y coordinates change by at most +/- 1, so that an up/down counter is sufficient for computing the next X and Y values.

Every time a valid point is encountered, a refresh cycle is performed and the trace algorithm halts for one cycle. The current row address is taken from the refresh counter and the refresh counter is incremented. Although every row is written the cycle after it is read, the refresh cycles guarantee that data will not be lost. The image is raster scanned in direction 0 so that a new row is read every cycle and proper refreshing will occur without special refresh cycles.

If all numbered test points are found to be in the off state, a point that has already been determined to be on the contour is checked. If this point is not on, the tracer will stop in an error state to indicate that the memory is not consistent.

| | | |
|---|---|---|
| 3 | 4 | 5 |
| 2 | * | 6 |
| 1 | 0 | 7 |

**Figure 1a. Direction Vectors from center point**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| x | * | 5 | 1 | x | * | 1 | x | x | 3 | 2 | 1 |
| x | * | 4 | 2 | * | 6 | 2 | * | * | 4 | * | x |
| 1 | 2 | 3 | 3 | 4 | 5 | 3 | 4 | 5 | 5 | 6 | * |
| | 0 | | | 1 | | | 2 | | | 3 | |
| 3 | 2 | 1 | 5 | 4 | 3 | 5 | 4 | 3 | * | 6 | 5 |
| 4 | * | x | 6 | * | 2 | * | * | 2 | x | * | 4 |
| 5 | * | x | * | x | 1 | x | x | 1 | 1 | 2 | 3 |
| | 4 | | | 5 | | | 6 | | | 7 | |

**Figure 1b. Search paths for each direction (0-7) between last 2 points**

### 7.6.3 HARDWARE

The hardware consists of four main components. These include the image RAM, the trace controller (FSM and clock generator), auxiliary counters and the clock generators (figure 2).

This circuit is quite different from many of the other image processing circuits that have been developed, in that a decision is made every cycle. This means that pipelining can not be used to the extent that it is in circuits with no decision making. In fact, the circuit is one large loop containing the FSM, the 16K RAM, and the counters. For correct operation of this loop, there must be exactly one delay from any point to itself. These facts dictate that the delay through all the components must be less than one clock cycle time. There was very little chance of designing these circuits for a 10 MHz clock rate in a 4μ NMOS technology. However, this does not mean that real-time operation is not possible with the chosen architecture. Because the circuit operates on the data a frame at a time, the algorithm must complete before one frame time (1/30 of a second) to achieve real-time operation. Therefore, the clock

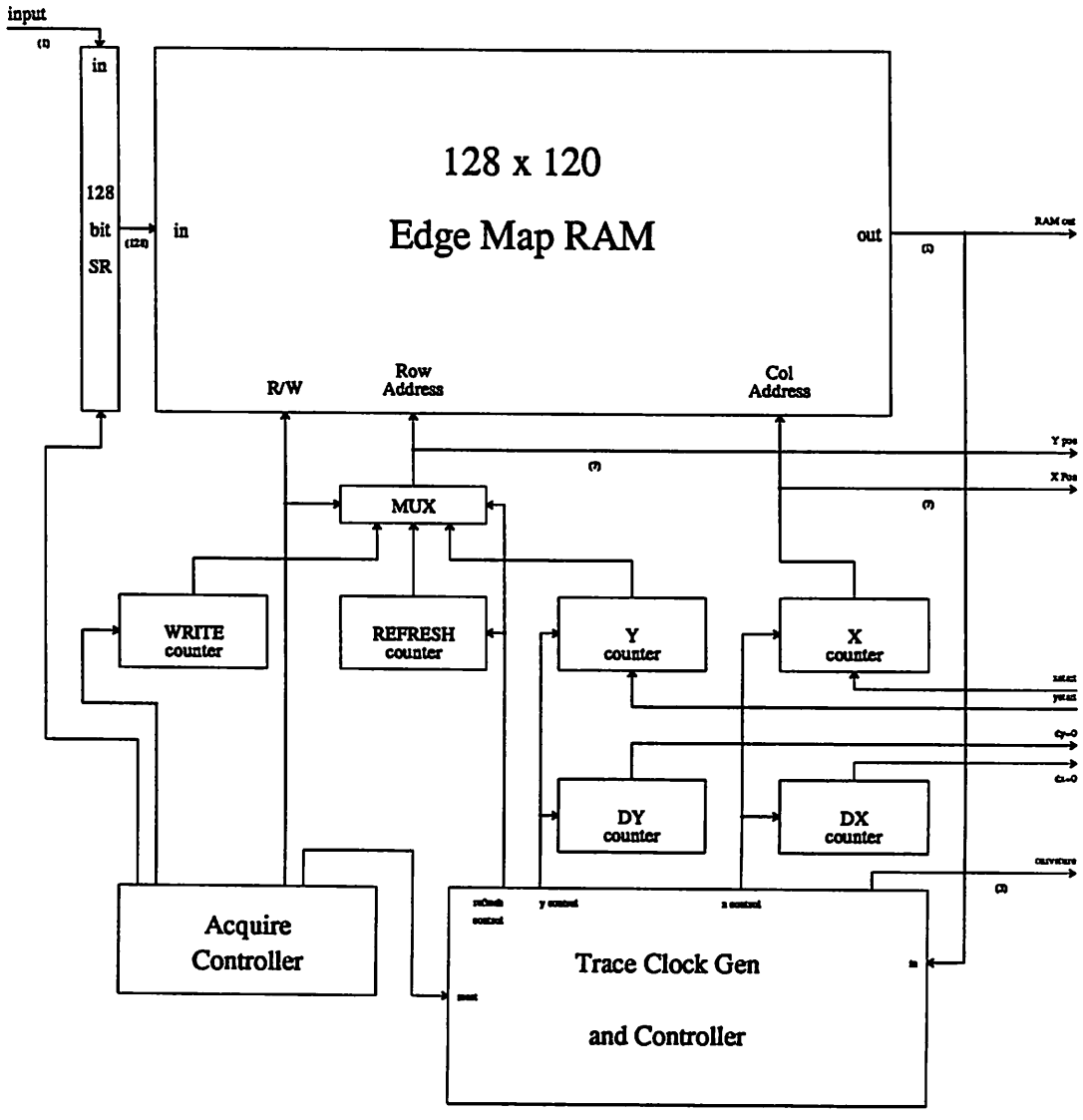



**Figure 2. Tracer Architecture**

rate can be reduced and still meet the real-time condition. Even if the algorithm required 2 frames for completion, two contour tracers could be used, each tracing objects in every other frame, to achieve real-time operation.

### 7.6.3.1 The RAM

The RAM stores the down-sampled edge map and is organized as an 128x120 array of three transistor cells. The 3T cell was chosen for the same reasons that it was used in the line delays. Separate read and write bit-lines and select lines makes it possible to read and write data in the same cycle with very few timing constraints.

When writing, the input is shifted into a 128-bit shift register during the horizontal display time and written into the array during the horizontal blank time one row at a time. In this way, the RAM-write operation is made very non-critical because the horizontal blank time is approximately 12 micro-seconds. Further, no decoder and other logic is needed to make it possible to write a single column at a time. When acquiring the image, the line number (Y) is supplied to the row decoder.

During RAM-read operations, the Y address of the test pixel is applied to the row decoder and the X address of the test pixel to the column decoder. To refresh the dynamic 3-transistor cell, every row read is automatically written back the next cycle. To ensure that all rows are refreshed, a refresh counter accesses rows sequentially at certain points in the tracing algorithm.

#### 7.6.3.2 The Trace Controller

The trace controller is made up of a FSM that implements the contour tracing algorithm and a clock generator.

The FSM consists of a ROM and a state register. For the tracing algorithm, there are 8 possible directions to travel in the array. For each of these 8 directions, there are 8 more possible directions to look for the next pixel. This makes a total of 64 states. However, some states are illegal, corresponding to the points marked with an "x", and can be used for refreshing, scanning and error trapping. 52 states are used for the tracing, 8 for refreshing, 3 for scanning and 1 for errors. The states are represented in 6 bits. The current direction is encoded in the top three bits of the state vector and the test direction is encoded in the lower three bits. In this way it is simple to determine what the tracer is doing from the state information.

The next state is computed solely from the current state and the image (1 bit) at the current point. Therefore, the only inputs to FSM ROM are the state and the RAM output. The outputs of the 128 x 16 ROM include the next state, $\Delta X$ and $\Delta Y$ to control the X,Y,DX and DY counters and the discrete curvature. To keep the magnitude of the curvature to a minimum, values corresponding to greater than $\pi$ (4-7) are translated to values less than 0 (-1 to -4). It can be shown that the curvature is a function of the current state and the RAM output and hence is easily looked up in the ROM.

The finite-state-machine approach was chosen to simplify the circuit design. Although the next state can usually be computed in a fairly straight-forward manner from the current state and the RAM output, a few special cases make the use of random logic more complicated and error prone. Further, using ROMs or PLAs for logic makes the circuits much easier to handle with CAD programs and simplifies changes. The FSM ROM was generated automatically with a ROM generator.

#### 7.6.3.3 The Counters

The current X and Y address, the current line being written, the refresh address, and the total distances traveled in X and Y (DX and DY) are kept in up/down counters that are controlled by the FSM. The use of separate counters was chosen to minimize the size of the FSM ROM. If the X and Y positions were included in the state of the FSM, the ROM would have become unreasonably large.

The total distance traveled in X and Y is computed so that it is possible to determine when the entire contour has been traced. When every pixel in the contour has been traced and the X and Y counters are at their original values, the DX and DY counters will both contain zero, which is detected and sent off-chip.

#### 7.6.3.4 Outputs

There are many signals sent off-chip for either testing and "debugging" or for actual use. The primary output is the curvature. The X and Y values are sent off-chip for computing the area, bounding box and moments of the contour. The next state is output for detecting when the tracer is raster scanning, computing the area in the feature extractor and for general testing. The DX=0 and DY=0 signals are used to detect when the contour has been traced. The carry out of the X and Y counters (CoutX and CoutY) is output to detect tracing off the edge of the frame and when the frame is done. CoutY occurs when the contour contacts the edge of the frame during tracing. CoutX during trace indicates that the contour is off the edge, while CoutX during scan indicates that the entire frame has been searched. The RAM output and the shift register output are used for testing.

#### 7.6.3.5 Inputs

The X and Y counters and state register can all be loaded with arbitrary values. The X and Y counters need to be loaded with arbitrary values to allow searching from various points in the frame so that multiple images will be found and traced. The state register only needs to be loaded with a single value (scanning state) in normal use, but was made fully loadable for testing. Other inputs include the video input, acquire signal, load control, and clocks.

#### 7.6.3.6 Testing

Since the chip is a FSM and all components are in the decision making loop, it was very important to be able to break the loop for testing. Otherwise, had the circuit not worked, it would have been difficult to control the FSM and isolate the problems. The RAM output is a very convenient place to break the feedback loop as it is only one bit wide. By allowing external signals to be substituted for the actual RAM output and by bringing the RAM output off chip, testing was made much simpler by making it possible to test parts of the chip independently.

#### 7.6.3.7 Clock Generators and Circuit Timing

There are two sets of low frequency clocks that are generated from 2 sets of high frequency clocks. The acquire (load array with image) and trace modes work with entirely separate clocks to allow asynchronous operation. The write controller generates clocks to latch the proper number of pixels into the shift register and to perform the down sampling (1/4) and is a circuit identical to the line delay controller with the OR plane programmed differently.

The trace clock generator (figure 3) generates the clocks (figure 4) for the trace modes. The primary reason for having the trace mode operate at a lower rate (1/8 input clock) is that the FSM approach does not allow pipelining and the critical path includes virtually the entire circuit. Also, having a lower clock rate makes it possible to generate more complicated signals for writing the RAM and pre-charging (the pre-charge signal is the same as slow phi2) because one slow clock cycle is 8 input clock cycles. It is also possible to get clock separation needed for RAM writing that is defined by the fast clock instead of by circuit parasitics.

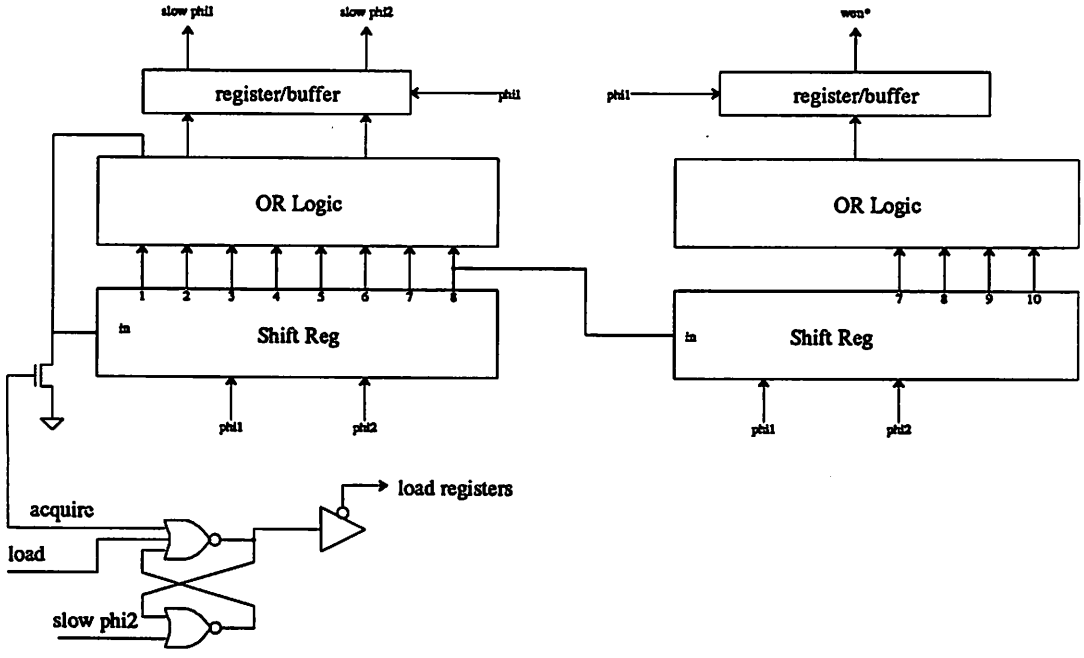



**Figure 3. Trace Clock Generator**

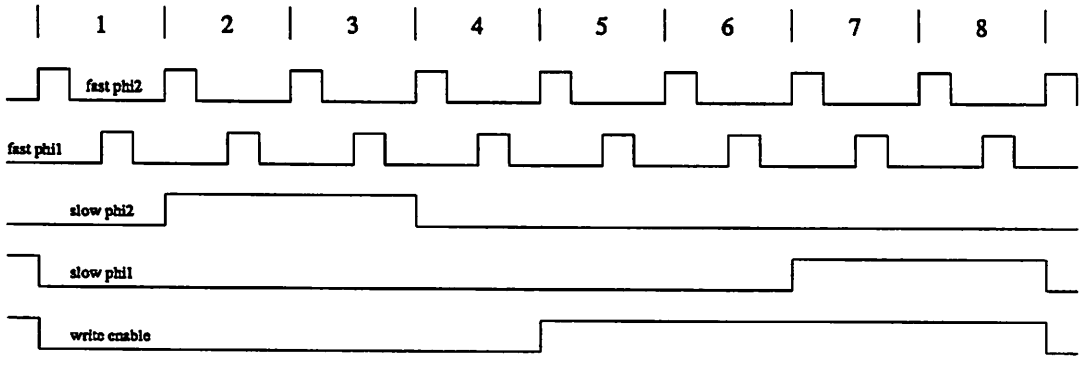



**Figure 4. Trace Clocks**

When acquiring a new image, the trace controller is held inactive and starts up in an orderly way after the image has been acquired. To prevent writing the wrong data into the RAM, refresh operations must be inhibited until valid addresses are available (figure 5).

### 7.6.4 IMPROVING CIRCUIT SPEED

The critical path in the tracer, during trace mode, starts at the X and Y counters and goes through the RAM, ROM and the ripple carry circuits of the X and Y counters. To improve the performance of the circuit, the ROM was organized as 64 words x 32 bits with a one bit column decoder. The RAM

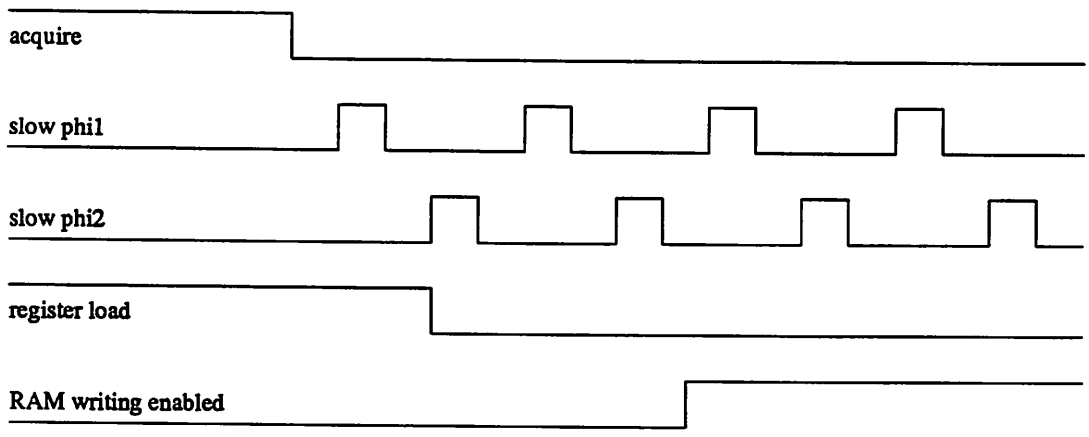



**Figure 5. Trace Clock Waveforms After Acquire**

output was chosen to be the column address. With this configuration, the row decoder and bit lines of the ROM can be settling while the RAM is being accessed. The RAM output only propagates through the column decoder and output circuitry of the ROM. The delay through that part of the ROM is quite small.

The up/down counters used in the circuit have identical slices and therefore have two gate delays per bit in the carry chain. Also, in the worst case, all carries are rising which is slow. To improve the speed of the counters even and odd up/down counter slices should be used.

### 7.6.5 REFERENCES

[1] Kitchin, P. W., et. al., "Processing of Binary Images," *Robot Vision*, IFS Ltd., UK, 1983.

[2] Pratt, W. K., *Digital Image Processing*, Wiley, New York, 1978, pp 543.

# 7.7 A Feature Extractor for the Image Contour Tracer

## 7.7.1 INTRODUCTION

A chip has been developed to interface the contour tracing chip to a host computer and to compute object features as the object is being traced. The center or mass, area, perimeter and bounding box of the image are computed. The chip also controls the loading of curvature data into an external buffer and reports the tracer status to the host.

The contour tracing chip only stores the image and implements the tracing algorithm. All data is generated at a high rate and can not be easily read by the slow host computer. Further, it has no provisions for indicating if the trace was finished successfully or with an error or if the entire frame has been scanned.

The feature extractor keeps track of the state of the contour tracer and indicates the state to the host processor. It also handles the loading of the curvature data into a buffer for reading by the host processor. Addresses, data and strobes are generated which can be used to load the data into an external RAM. This chip generates the starting coordinates for the next search. For schemes not requiring the more complete information contained in the curvature, basic features of the image are computed by the feature extractor and are available immediately after the tracing is completed.

The system diagram showing the inter-connection of the tracer, the feature extractor and the host computer is shown in figure 1. The feature extractor loads the state and X and Y coordinate outputs of the tracer over an input bus. There is a common I/O bus that is used to read results from the feature extractor by the host, load constants from the host into the feature extractor, load start values from the feature extractor into the tracer and set the address of the external buffer RAM from the extractor during tracing or from the host. The curvature data passes through a second bus to the RAM or from the RAM to the

host. Finally, there are control signals that set the feature extractor operation and status outputs to signal the host that the tracer has finished its current operation.

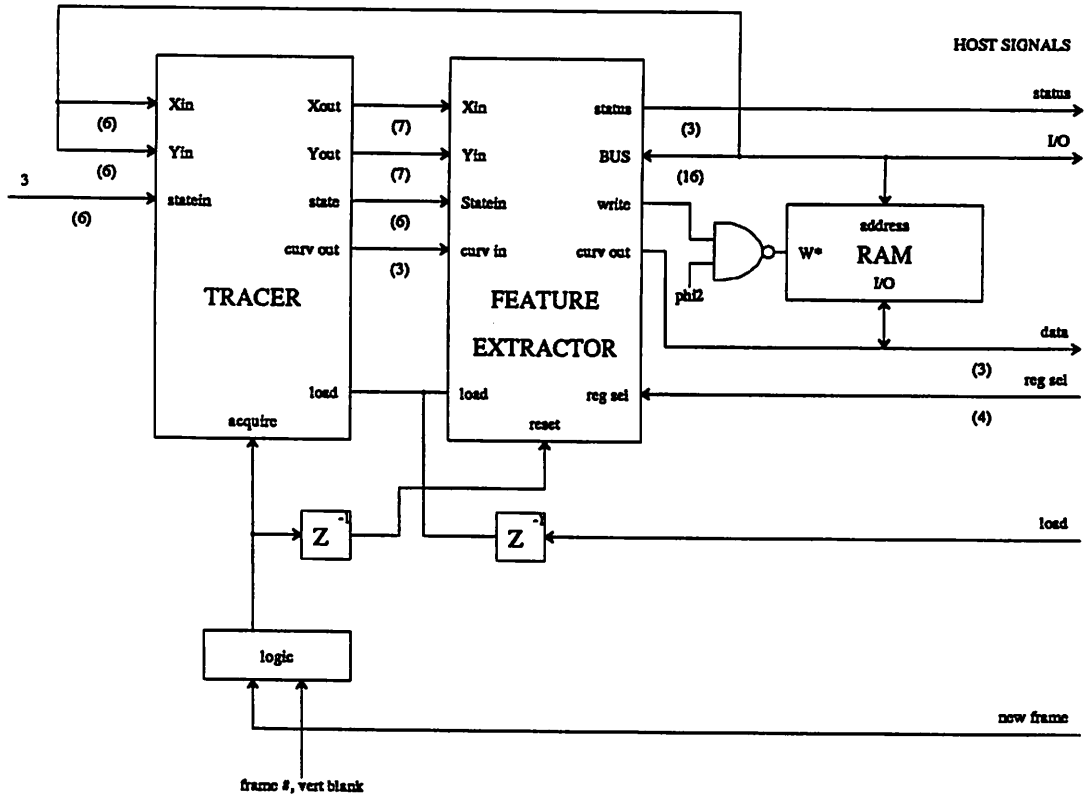



**Figure 1. System Organization**

The feature extractor-tracer combination operates in several distinct modes. During the acquire mode, the image is loaded into the tracer RAM and the feature extractor is idle except that the user chosen start position is on the I/O bus to be loaded into the tracer X and Y registers. When the image has been acquired, the tracer loads the start position from the I/O bus and begins searching for an image. While the tracer is searching for an image, the feature extractor resets all internal registers for the next feature computations. After the image is found, the feature extractor puts the RAM address on the I/O bus for loading data into the RAM and begins computing features from the X,Y and state outputs of the tracer. When the contour tracer finishes tracing (either successfully or with an error) the host processor is notified via a feature extractor output. The tracer continues tracing the last image while the feature extractor is idle. At this point the host can load the computed features from the feature extractor and the curvature data from the external RAM. The feature extractor then puts the new start position on the I/O bus for the next trace. When the host resets the tracer, by loading the search state into the state register,

the feature extractor again computes features and signals the host when the trace is done. If the search continues and no image is found, the feature extractor notifies the host that the entire frame has been searched. The host can then repeat the whole process by starting another acquire operation.

### 7.7.2 COMPUTED FEATURES

The features computed include: the perimeter, the even perimeter, the area, the center of mass (except for the required division) and the bounding box. The perimeter is computed by counting the number of valid points on the contour. The perimeter is also used as the address for the buffer RAM for loading the curvature. The even perimeter is the number of contour points where the direction had an even number (i.e. the tracer did not take a diagonal step) and is computed so that an estimate of the true perimeter can be determined, since the diagonal steps are longer than the vertical or horizontal steps. The bounding box is computed by keeping track of the minimum and maximum values of both the X and Y coordinates. The center of mass of the contour is computed as opposed to the center of mass for the solid figure. That is, the image is treated as though the center of the image has no mass. To obtain the center of mass of the contour, the sums all the X coordinates (xbar) and all the Y coordinates (ybar) are divided by the perimeter.

The area is determined as the image is traced by accumulating the area under the contour when tracing in one direction and subtracting the area under the contour when tracing in the other direction. The trapezoidal approximation for integration is computed and is derived below:

$$area = \sum_i \frac{x_i + x_{i-1}}{2} \Delta y_i$$

but $x_{i-1} = x_i - \Delta x_i$

$$\text{so } area = \sum_i \frac{x_i + x_i - \Delta x_i}{2} \Delta y_i = \sum_i x_i a_i + b_i$$

where $a_i = \Delta y_i = \{-1, 0, +1\}$ (determined from tracer direction)

$$\text{and } b_i = \frac{\Delta x_i \Delta y_i}{2} = \{-1/2, 0, 1/2\} \text{ (determined from tracer direction)}$$

It can be seen that the area can be computed by performing two sums and a simple multiply. A

PLA is used to compute the values $a_i$ and $b_i$ from the current tracer direction.

In order to simplify the feature extractor, some operations are left to the host processor to perform. The division and multiplication operations were not implemented on-chip because these operations require a relatively large area and would be needed only once per contour trace. Therefore, to compute the center of mass, the host processor would load the sums of the X and Y coordinates and the perimeter and perform the division. The true perimeter would be obtained similarly. The host would load the perimeter and the even perimeter values from the feature extractor and perform the final multiplication and addition. By leaving the multiplication and division operations off chip, all features could be computed with adders, comparators and incrementers which are all quite small.

### 7.7.3 THE HARDWARE

The chip (figure 2) consists of an arithmetic unit that is composed of three sub-arithmetic units that compute the features of the current image and a control section consisting of 2 PLAs and a finite-state-machine (FSM). There are two 7-bit input data busses for the X and Y coordinates. A single 16-bit I/O bus is used to send results to the host processor and to load constants from the host processor. The 16-bit bus also transmits the address to the buffer RAM and the starting coordinates to the contour tracer. Control signal inputs include the 6-bit contour-tracer state and the 3-bit tracer output. The features are computed in parallel processors (figure 3), one simple processor computing each feature.

The perimeter processors are simply counters which can be cleared. The xbar, ybar processors are accumulators that can be reset. The bounding box processors are resetable maximum or minimum processors. The area computation requires two adders and a complimenter. All registers except the minimum and maximum registers will saturate.

#### 7.7.3.1 General Organization

The arithmetic functions were spread over three units to accommodate varying requirements for data word width. The bounding box processors (min-max circuits) only need a width equal to that of the input data width (7 bits). The perimeter, area, and moment processor require more bits. The theoretical maximum limits for a 128x120 image are:

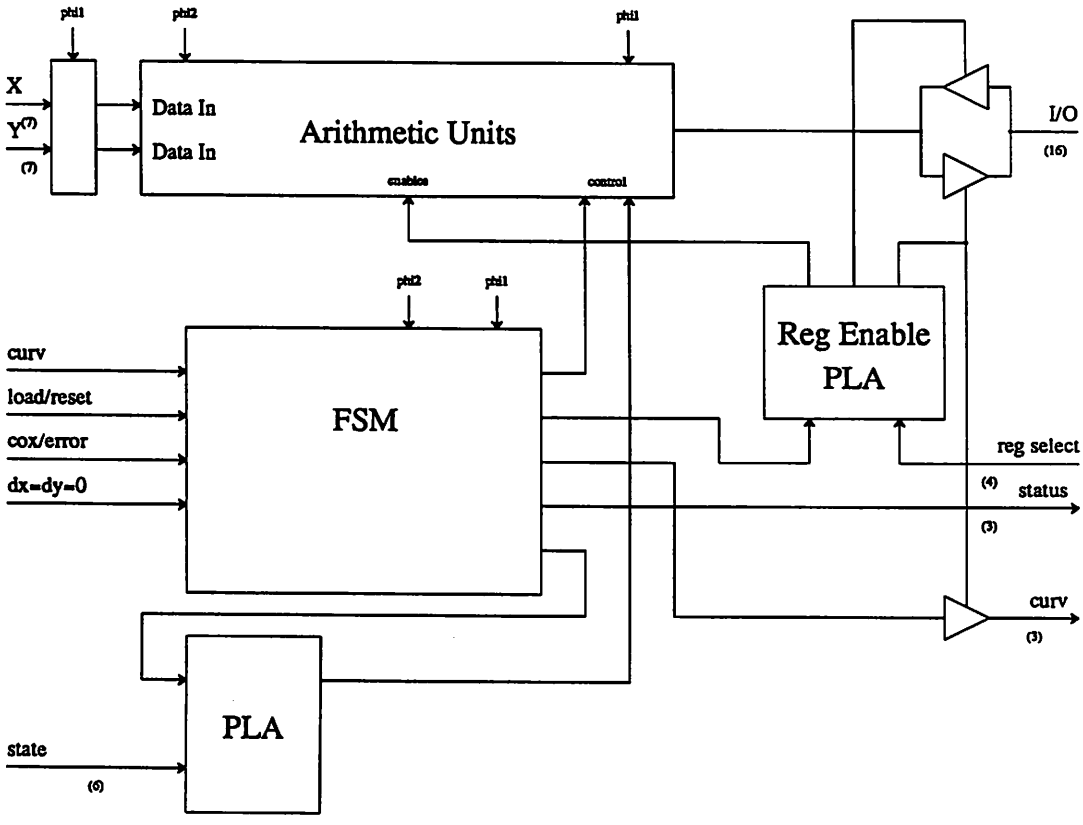



**Figure 2. Chip Architecture**

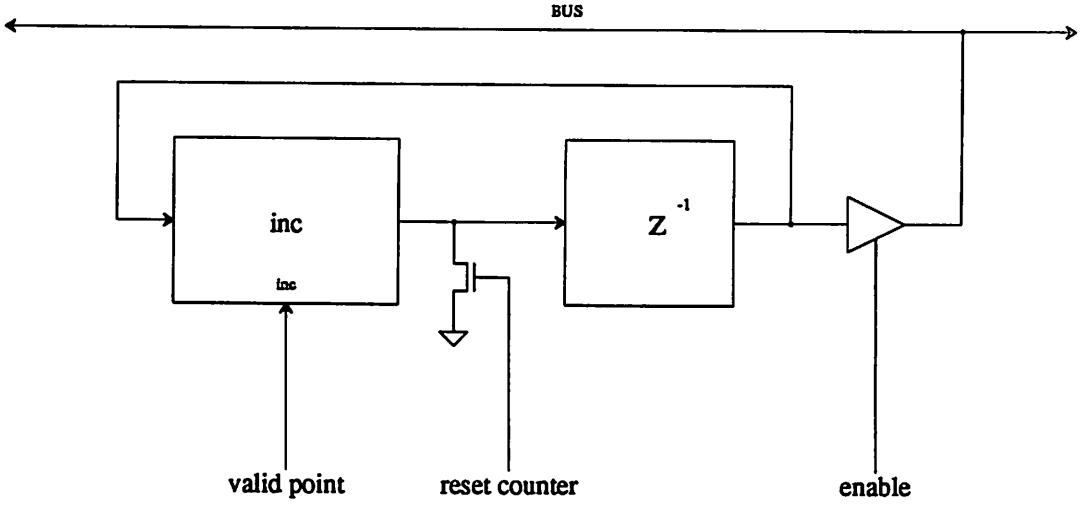



**Figure 3a. Processor for Perimeter and Even Perimeter**

area: 128x120 or 14 bits
xbar: 23 bits
ybar: 23 bits
perimeter: 14 bits
even perimeter: 14 bits

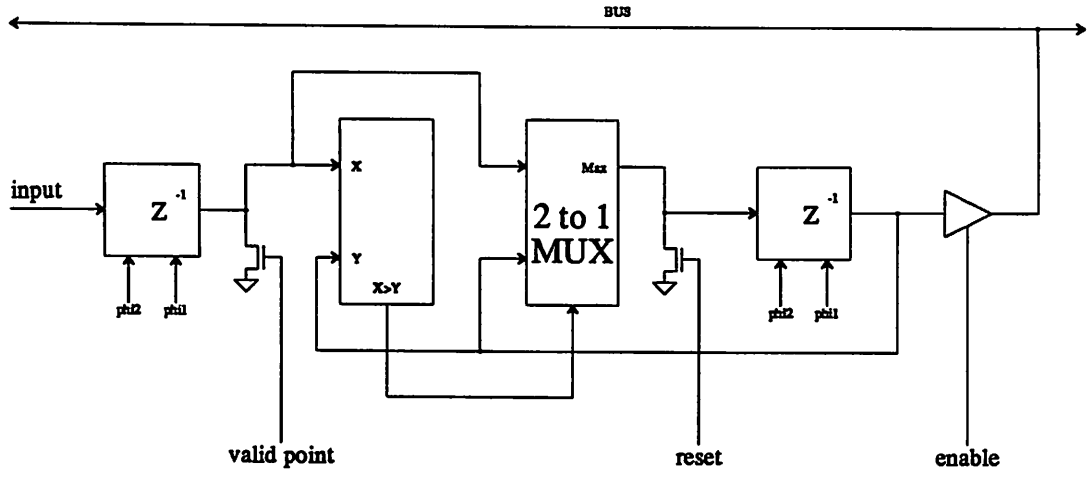



**Figure 3b. Processor for Y and X Maximum**

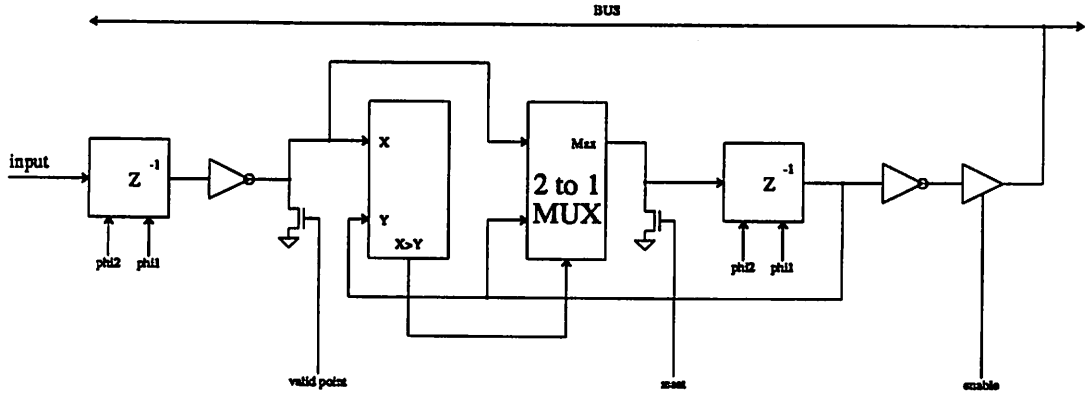



**Figure 3c. Processor for Y and X Minimum**

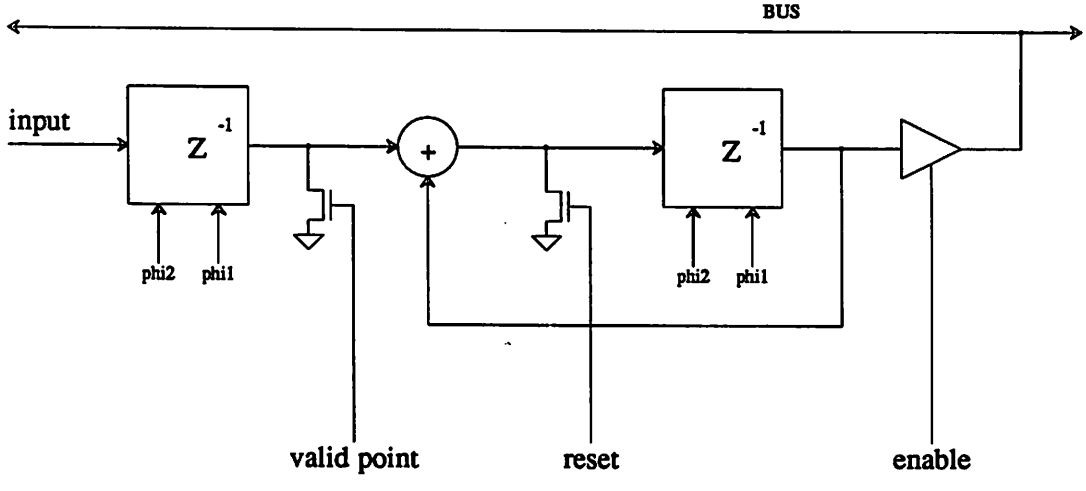



**Figure 3d. Processor for Center of Mass**

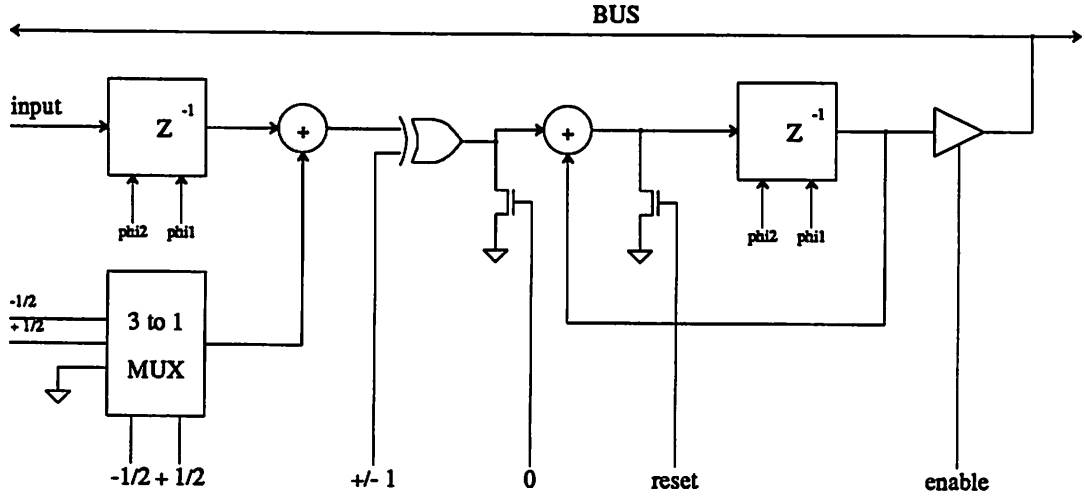



**Figure 3e. Processor for Area**

The processor for these quantities was chosen to be 16 bits for several reasons. First, the limitation to 64 pins on the chip was a factor. Second, it is a convenient number for connecting to the host computer (usually I/O ports are multiples of 8 bits). Finally, although the xbar and ybar computations may require more bits, only extremely complicated non-convex shapes will required more than 16 bits. For these complex shapes, it is unlikely that the center of mass will be very useful for determining the position of the object as the center of mass will be a strong function of the quantization.

### 7.7.3.2 Control

The control functions of this chip are very data dependent (the tracer state data) so an FSM is employed to handle the required decision making. One PLA and the FSM determine of the contour tracer state and generate control signals for the arithmetic processors. The PLA generates all signals which are a function of the contour tracer state (detects when the tracer is scanning and generates $a_i$ and $b_i$ for the area computations). The FSM also performs unrelated functions to decrease the number of circuit blocks needed on chip. For example, the delays needed to synchronize the curvature signals with control signals for writing into the buffer RAM are provided by the FSM. This is not an efficient use of the FSM but avoids the need for random wiring to additional circuit blocks.

The state transition table for the FSM is given in table 2. When the tracer is acquiring an image, the FSM is forced to the acquire state via the reset signal. When reset goes low, the FSM goes to a waiting

state. It will wait there until it is detected that the tracer is scanning. When scanning is detected, the FSM will go to the scan state. If scanning is not detected, an error will be flagged because the tracer started on a contour and correct characterization is not possible. The FSM will wait in the scan state until either the end of frame is reached (*COX* true) or a contour is found (*scanning* false). If a contour is found, the FSM goes to tracing state where it remains until the trace is done or an error occurs. The FSM waits in the error state and the contour done state until either a *LOAD* signal is encountered to reset the tracer at a new point or *RESET* occurs to acquire a new image. The *resetreg* signal goes high during the waiting state to reset all processors for new computations. *Selstart* goes low during tracing to allow for the RAM address to be put on the I/O bus instead of the tracer start position.

Another PLA handles the register selection and permits asynchronous access of the internal registers by the host computer. If the user sets the override signal high, then the registers can be selected (enabled onto the 16-bit I/O bus) by the three address lines for reading by the host processor. In this mode the circuit looks to the host processor like a 7-word static ram. If override is low, the feature extractor along with the user supplied register address determines which register is enabled. Registers are enabled to provide the start position for the tracer while the tracer is searching and the address for writing data into the RAM while the tracer is tracing. The register address inputs give the user control over the choice of start positions for the tracer to begin searching for a new contour after one or more contours have been found and traced. Register control information is given table 1.

### 7.7.4 USE OF AUTOMATED LAYOUT TOOLS

Almost the entire circuit layout was generated automatically. The three data-paths were assembled with the data-path generator and the 2 PLAs and the FSM were assembled with Modgen. The resulting blocks were, however, placed and routed by hand. The automated aspects of this circuit made changes quite easy. Several mistakes were found that could be quickly corrected by changing the circuit description file. Some changes to the FSM did not even change the block size and hence did not require any routing changes at the chip level.

### 7.7.5 TESTING

This circuit was fairly easy to test because there are no large arrays of memory and the outputs of any processor can be watched during computation. The user can force any processor to put its output on the I/O bus at any time as described previously. That makes it possible to observe the partial results during all phases of operation. The FSM state is brought off-chip in encoded form which is quite useful for testing to verify that the state is determined properly.

| *override* | *RESET** | *selstart** | address | bus direction | bus data |
|---|---|---|---|---|---|
| 1 | x | x | 0 | to host | ymax, xmax |
| 1 | x | x | 1 | to host | count of pixels around contour |
| 1 | x | x | 2 | to host | count of non-diagonal pixels |
| 1 | x | x | 3 | to host | xmin, xmin |
| 1 | x | x | 4 | to host | ybar |
| 1 | x | x | 5 | to host | xbar |
| 1 | x | x | 6 | to host | area |
| 1 | x | x | 7 | from host | ystart, xstart |
| 0 | 1 | x | x | to host | ystart, xstart |
| 0 | 0 | 1 | 0 | to host | ystart, xstart |
| 0 | 0 | 1 | 1 | to host | ymax, xstart |
| 0 | 0 | 1 | 2 | to host | ystart, xmax |
| 0 | 0 | 1 | 3 | to host | ymax, xmax |
| 0 | 0 | 0 | x | to host | pixel count (RAM address) |

* - internal signals

**Table 1. Register Control**

| current state | | input | next state | done | out code | rr | ss |
|---|---|---|---|---|---|---|---|
| name | # | | | | | | |
| scan | 000 | $scanning*\overline{COX}*\overline{RESET}$ | 000 | 0 | 00 | 0 | 1 |
| | | $scanning*\overline{RESET}$ | 001 | 0 | 00 | 0 | 0 |
| | | $scanning*COX*\overline{RESET}$ | 100 | 0 | 00 | 0 | 1 |
| tracing | 001 | $DX=DY=0*\overline{ERROR\ 1}*\overline{RESET}$ | 010 | 0 | 01 | 0 | 0 |
| | | $\overline{ERROR\ 1}*\overline{RESET}$ | 011 | 0 | 01 | 0 | 0 |
| | | $\overline{ERROR\ 1}*\overline{DX=DY=0}*\overline{RESET}$ | 001 | 0 | 01 | 0 | 0 |
| contour done | 010 | $\overline{\overline{RESET}*LOAD}$ | 110 | 1 | 00 | 0 | 1 |
| | | $\overline{RESET}*\overline{LOAD}$ | 010 | 1 | 00 | 0 | 1 |
| error | 011 | $\overline{\overline{RESET}*LOAD}$ | 110 | 1 | 01 | 0 | 1 |
| | | $\overline{RESET}*\overline{LOAD}$ | 011 | 1 | 01 | 0 | 1 |
| frame done | 100 | $\overline{RESET}$ | 100 | 1 | 11 | 0 | 1 |
| acquire | 101 | $\overline{RESET}$ | 110 | 0 | 11 | 0 | 1 |
| waiting | 110 | $scanning*\overline{RESET}*\overline{LOAD}$ | 000 | 0 | 10 | 1 | 1 |
| | | $\overline{scanning}*RESET*\overline{LOAD}$ | 110 | 1 | 10 | 1 | 1 |
| | | $\overline{RESET}*LOAD$ | 110 | 0 | 10 | 1 | 1 |
| | xxx | *RESET* | 101 | 0 | 11 | 0 | 1 |

rr = resetreg, ss = selstart

**Table 2. State Transition Table**

$scanning = (tracerstate=3)$

$ERROR\ 1 = \overline{COX}+\overline{COY}+\overline{ERROR}$

$validpoint = (out\,0+out\,1+out\,2+(tracerstate=36))*(tracing)$

# 7.8 The Line Delay

## 7.8.1 INTRODUCTION

To perform two-dimensional signal processing algorithms on the one-dimensional video signal, line delays are required. The basic function required of the line delays is to accept the current pixel value and output the value of the pixel "directly above" the current pixel (i.e. delayed by one video line).

## 7.8.2 HARDWARE CHOICES

The line delay could be implemented with a shift register that has a length equal to the number of pixels in the line. This would be a simple solution, but would require a significant amount of space and power. To avoid these drawbacks, a RAM is used and a pointer to the data is shifted instead of the data. Since every location in the RAM is written at the video line rate (i.e. every 75 micro-seconds), a dynamic memory with no refresh can be used. By using the three transistor RAM (3T cell) with separate read and write selects and separate read and write bit lines, a read and a write operation can occur in the same cycle with both operations having the full cycle time for completion. The 3T cell with separate bit lines is only slightly larger than that with a single bit line, so a high price is not paid for the performance increase.

Figure 1 shows the basic operation of the RAM array. The shift register pointer is at the bottom of the diagram. The shift register is currently enabling the left cell for writing and the right cell for reading. Input data on the write bit line is written into the left cell, while data stored in the right cell is put on the read bit line and output. In the next cycle, the pointer will shift to the right and the cell that is currently being read will be written.

The shift register pointer replaces a row decoder and a counter. Although the total number of transistors is virtually the same for both cases, the circuit with a shift register is more compact because

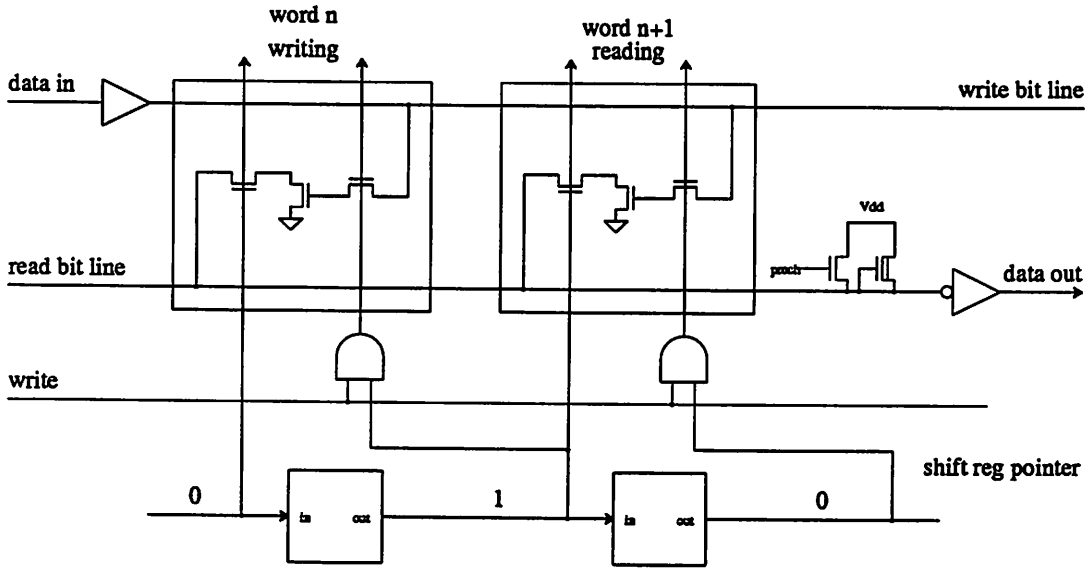



**Figure 1. 3 Transistor Cell Operation**

the shift register abuts the RAM array while the counter must be a separate block and routed to the RAM decoder.

The fact that the data is always written and read in the same order can be exploited to ease the circuit design with some increase in architecture complexity (figure 2). The input data is de-multiplexed (by a factor of eight) to form a 64-bit word that changes at a 1.25 MHz rate. The internal RAM array works at one-eighth speed, making low power implementations feasible. The array is also square, 64 rows by 64 columns for 512 pixels per line with 8-bit data, which is easier to handle when putting the entire chip together. At the RAM output, the 64-bit, 1.25 MHz data is multiplexed to form the desired 10 MHz 8-bit word.

The de-multiplexor consists of an 8-bit recirculating shift register, a set of 8 8-bit registers (there are actually only 7 registers, but to simplify the discussion it will be assumed that there are 8 registers) and a 64-bit register. The set of 8 registers latches 8 consecutive pixels at the video rate (10 Mhz for 512 pixel lines). The shift register has only one output high and provides the load signals for the registers. On the first cycle, the shift register points to the first (top most) register and the first register latches a pixel value. The next cycle (10 Mhz cycle), the shift register pointer shifts and the $2^{nd}$ pixel is latched into the $2^{nd}$ register. After 8 cycles, the 8 pixels are latched into the 8 registers forming a 64-bit word. To synchronize this data with the RAM which is running a one eighth speed, the 64 bits of data are

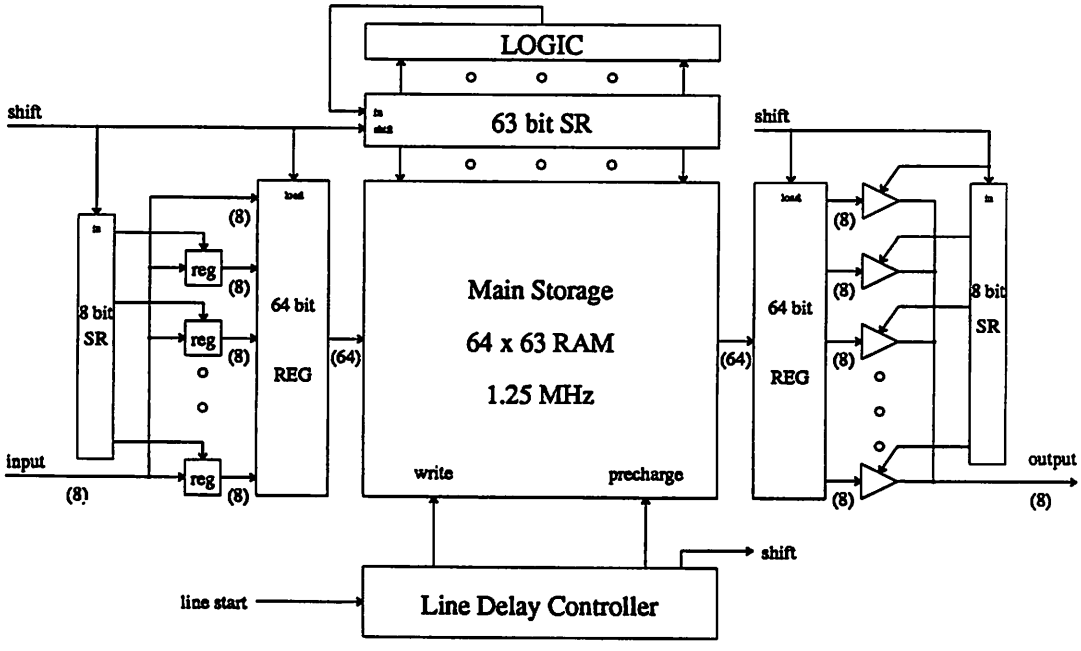



**Figure 2. Line Delay Architecture**

latched into the 64-bit register. At the output of this register, all 64 bits change at the same time unlike the outputs of the first set of registers. The data is then written into the RAM.

The first 8-bit register does not really exist and is only shown for the sake of simplicity. It is not needed because the multiplexor adds an extra delay of 8 pixels to the top most path of the circuit. By removing the top most input latch, the pixel is advanced by 8 cycles in the de-multiplexor, canceling the delay in the multiplexor. Therefore, the first pixel is latched directly into the top 8 bits of the 64 bit register. The other 56 bits come from the 7 8-bit registers.

The multiplexor operates similarly. A 64-bit register latches (at the lower rate) the RAM output which consists of 8 consecutive pixels. A shift register pointer, enables the 8-bit tri-state buffers in succession at the high rate. The output bus then contains the 10 MHz 8-bit signal that has been delayed by one video line.

The operation of the line delay is shown in tables 1a-f. Each table shows the register contents at a multiple of 8 cycles. The input and output columns indicate the 8 inputs and outputs that occurred during those 8 cycles (the top entries are the first inputs and outputs). Two rows of the RAM array are also shown. In table 1a, the first 8 pixels have been latched. Note that pixel 1 was latched directly into the

64-bit register. 8 cycles later (table 1b), pixels 9-16 have been latched and the 64 bit register was written into the RAM array. After another 8 cycles (table 1c), pixels 2-9 are written into the RAM array. Table 1d shows the situation at the very end of the video line. Pixels 505-512 (last 8 pixels on the line) are latched into the input registers and $row_n$ is read from the array and latched into the output register. As mentioned, the top most pixel in the output latched is delayed 8 cycles and does not appear at the outputs until pixel 513 is at the input. After $row_n$ has been read, it is over written with the data in the input register. At the same time, $row_{n+1}$ is read and latched in the output register and multiplexed on the output lines. The data appearing at the output is in the proper order and exactly 512 pixels behind the input.

Without this de-multiplexing scheme and with a 4μ NMOS technology at 10 MHz, large buffers would be needed to drive the select lines, particularly the write select line. To prevent the wrong data from being written into the cells, the write select must be gated with one clock phase (~40 nS) and must pull up close to $V_{DD}$ to get a good a high stored charge. Simulations showed that quite a bit of power would be dissipated and that 3μ technology would be needed since the RC delays were quite large.

The multiplexing and de-multiplexing logic requires a significant amount of space. If a more advanced technology were available, it would be possible to operate the RAM array at the data rate without the MUX-DEMUX circuitry and hence save the space associated with these circuits.

### 7.8.3 HARDWARE PARAMETERS

The primary parameters of the line delay are the line length ($L$) and the number of bits/pixel ($B$). The multiplexing and de-multiplexing ratios ($M$) are also important. The line delay can be easily modified to handle other line lengths, but the line length must be a multiple of M. The internal RAM should have $\frac{L}{M}-1$ rows. It does not have $\frac{L}{M}$ rows because the other registers account for a single slow cycle delay. The RAM should have $MB$ columns. Arbitrary word widths can not be handled because the shift registers which accomplish the latching of data at the input and the enabling of data at the output, are put between the data lines and must remain even if fewer bits are desired. The number of bits/word can vary by multiples of two, due to mirroring in the circuit, between 6 and an upper limit determined by timing constraints. For small numbers of bits, the general approach taken requires a lot of overhead and

may no longer be better than a direct implementation.

### 7.8.4 CONTROLLER

A controller is used to generate the lower frequency clocks for the line delay and to shift in exactly one line of video data. The controller (figure 3) consists of an 8 bit shift register (SR), an OR logic array, a counter and start-stop logic. The slow clocks required for the line delay are pre-charge, write enable and shift.

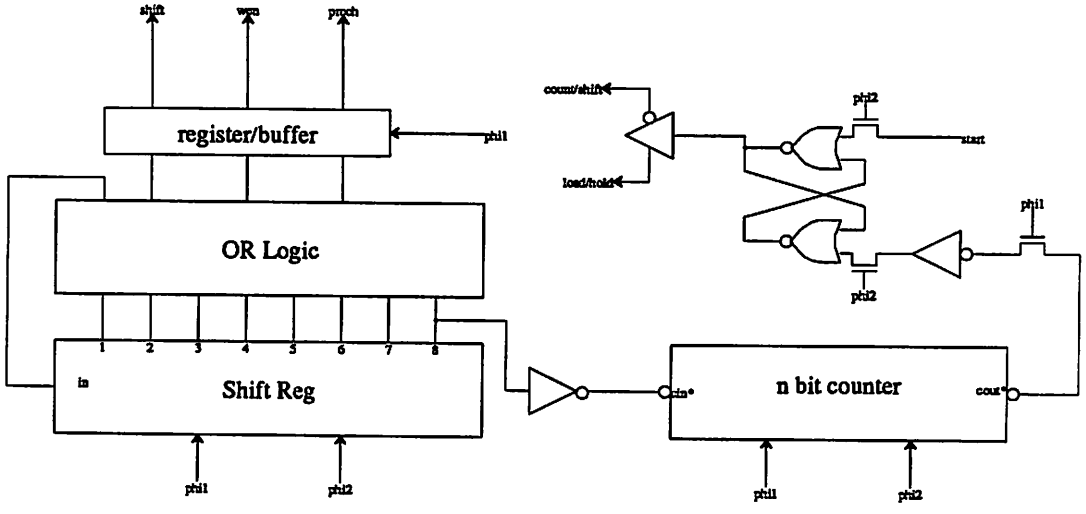



**Figure 3. Line Delay Controller**

It was desired to make the slow clocks be fully programmable. That is, each slow clock can go high during any combination of the 8 fast cycles that make one slow cycle. One way to implement this is to have a 3-bit counter that is connected to a PLA or ROM. The 8 states could then be decoded in an arbitrary manner. However, the counter and row decoder of the ROM can be replaced by an 8-bit SR. The shift register outputs are identical to the row select lines of the ROM. If the OR plane is connected to the output of the SR, the equivalent of a ROM has been made. This was the approach taken. One bit in the OR plane is used to ensure that the SR has only one output high at a given time and the rest of the bits control one slow clock each. To have a slow clock active during a particular fast clock cycle, a transistor is put in the OR array at the corresponding SR output.

To ensure that exactly one line of the video signal is shifted into the delay line, additional hardware is required. A counter keeps track of the total number of slow cycles that has occurred. When this total

reaches the number of slow cycles that occur in one video line (64 slow cycles occur in a 512 pixel line), the controller is shut down. The shift register stops shifting and the clocks are left in an inactive state. When the "start" (start of line) signal goes high, the controller starts again and shifts in another line. If the "start" remains high, the controller free runs which is useful for testing.

The slow clocks generated by the controller are shown in figure 4. Shift (same as "next 8 pixels" in figure 2) goes high once every 8 cycles. This causes the 63-bit shift register pointer to shift once every slow cycle and keeps exactly one output of each of the 8-bit shift register pointers active in any cycle. It also causes data to be latched into the 64-bit RAM input and output registers. The write enable goes low one cycle before shift and returns high one cycle after shift to disable write operations when the RAM addresses are changing. The bit lines are pre-charged high at the beginning of each new read cycle (for three cycles after shift goes low).

The controller stops at cycle 8 when waiting for the beginning of the next line. In this state, RAM write operations are disabled, the main shift register pointer is frozen and no pre-charging takes place. Basically, the circuit is inactive in cycle 8.

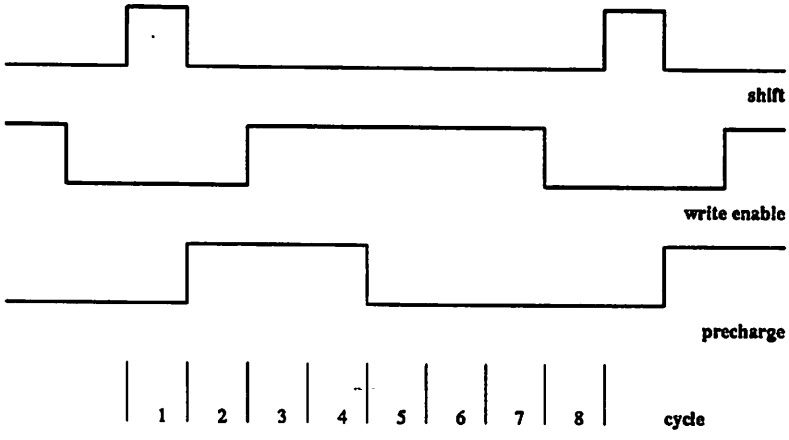



**Figure 4. Line Delay Clocks**

The controller is easily reconfigured for other line lengths (L) or number of fast cycles per slow cycles (R). The shift register length is equal to R. The counter must count up to L/R.

| input | reg1 | reg2 | $row_n$ | $row_{n+1}$ | outreg | output |
|---|---|---|---|---|---|---|
| 1 | | 1 | x | x | x | x |
| 2 | 2 | x | x | x | x | x |
| 3 | 3 | x | x | x | x | x |
| 4 | 4 | x | x | x | x | x |
| 5 | 5 | x | x | x | x | x |
| 6 | 6 | x | x | x | x | x |
| 7 | 7 | x | x | x | x | x |
| 8 | 8 | x | x | x | x | x |

| input | reg1 | reg2 | $row_n$ | $row_{n+1}$ | outreg | output |
|---|---|---|---|---|---|---|
| 9 | | 9 | 1 | x | x | x |
| 10 | 10 | 2 | x | x | x | x |
| 11 | 11 | 3 | x | x | x | x |
| 12 | 12 | 4 | x | x | x | x |
| 13 | 13 | 5 | x | x | x | x |
| 14 | 14 | 6 | x | x | x | x |
| 15 | 15 | 7 | x | x | x | x |
| 16 | 16 | 8 | x | x | x | x |

| input | reg1 | reg2 | $row_n$ | $row_{n+1}$ | outreg | output |
|---|---|---|---|---|---|---|
| 17 | | 17 | 1 | 9 | x | x |
| 18 | 18 | 10 | x | 2 | x | x |
| 19 | 19 | 11 | x | 3 | x | x |
| 20 | 20 | 12 | x | 4 | x | x |
| 21 | 21 | 13 | x | 5 | x | x |
| 22 | 22 | 14 | x | 6 | x | x |
| 23 | 23 | 15 | x | 7 | x | x |
| 24 | 24 | 16 | x | 8 | x | x |

| input | reg1 | reg2 | $row_n$ | $row_{n+1}$ | outreg | output |
|---|---|---|---|---|---|---|
| 505 | | 505 | 1 | 9 | 1 | x |
| 506 | 506 | 498 | x | 2 | x | x |
| 507 | 507 | 499 | x | 3 | x | x |
| 508 | 508 | 500 | x | 4 | x | x |
| 509 | 509 | 501 | x | 5 | x | x |
| 510 | 510 | 502 | x | 6 | x | x |
| 511 | 511 | 503 | x | 7 | x | x |
| 512 | 512 | 504 | x | 8 | x | x |

| input | reg1 | reg2 | $row_n$ | $row_{n+1}$ | outreg | output |
|---|---|---|---|---|---|---|
| 513 | | 513 | 505 | 9 | 9 | 1 |
| 514 | 514 | 506 | 498 | 2 | 2 | 2 |
| 515 | 515 | 507 | 499 | 3 | 3 | 3 |
| 516 | 516 | 508 | 500 | 4 | 4 | 4 |
| 517 | 517 | 509 | 501 | 5 | 5 | 5 |
| 518 | 518 | 510 | 502 | 6 | 6 | 6 |
| 519 | 519 | 511 | 503 | 7 | 7 | 7 |
| 520 | 520 | 512 | 504 | 8 | 8 | 8 |

| input | reg1 | reg2 | $row_n$ | $row_{n+1}$ | outreg | output |
|---|---|---|---|---|---|---|
| 521 | | 521 | 505 | 513 | 17 | 9 |
| 522 | 522 | 514 | 498 | 506 | 10 | 10 |
| 523 | 523 | 515 | 499 | 507 | 11 | 11 |
| 524 | 524 | 516 | 500 | 508 | 12 | 12 |
| 525 | 525 | 517 | 501 | 509 | 13 | 13 |
| 526 | 526 | 518 | 502 | 510 | 14 | 14 |
| 527 | 527 | 519 | 503 | 511 | 15 | 15 |
| 528 | 528 | 520 | 504 | 512 | 16 | 16 |

**Tables 1a-1f. Data Storage in the Line Delay is Shown over 512 cycles.**

# CHAPTER VIII RECOGNITION RESULTS AND ANALYSIS

## 8.1 INTRODUCTION

The image recognition system described in chapter 3 has been built using the custom chips which were described in chapter 7. The system accepts a standard 10 MHz video image and outputs features (the curvature, area, perimeter, center of mass and bounding box) at low rate for use by a host computer.

The feature based recognizer was implemented because the three basic features ($\frac{A}{P^2}$, $P\sigma^2_{curv}$ and $\sum_{curv>0} curv$) could be computed quickly. This made it possible to collect a large amount of data. Recognition based upon the Fourier descriptors and the curvature signal were examined to a lesser extent.

The overall system does not quite achieve real-time operation, that is, the recognition can not performed on all frames, because some feature extraction and the pattern matching is performed by the host computer. It would be possible to achieve real-time operation if the remaining feature extraction and pattern matching were performed by either a special purpose chip or by a faster general purpose processor. The system can recognize approximately 7 $\frac{frames}{Sec}$ when $P\sigma^2_{curv}$ and $\sum_{curv>0} curv$ must be computed on the SUN host computer and approximately 15 $\frac{frames}{Sec}$ when only the features computed by the feature extractor are used.

The results of the recognition process for a set of 8 simple objects (see figure 1) have been obtained for the system described in chapter 3. The basic objects were chosen to simplify the analysis of the results. Several very different geometric shapes were used in addition to some similar complex shapes. From the results with these objects it is possible to determine some of the strengths and weaknesses of the techniques used.

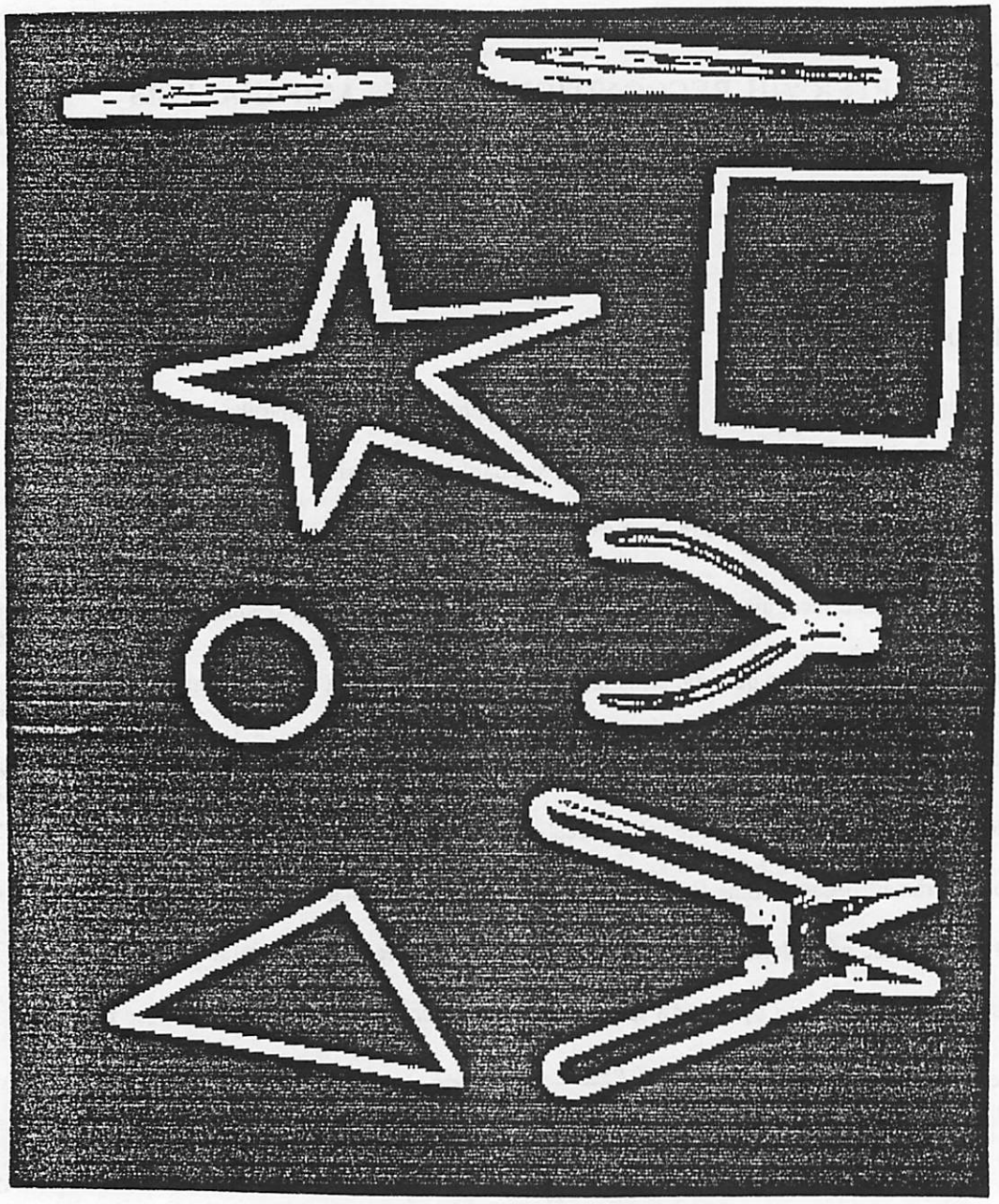


**Figure 1. Objects Used in Recognition Tests**

The recognition task was chosen as a good test of the custom circuits which have been designed and the basic algorithms chosen. For recognition problems with fewer degrees of freedom (e.g. size invariance is not required), it is possible to come up with more robust recognizers which require fewer computations.

## 8.2 THE FEATURES

In chapter 3, the ideal properties of the features were discussed. It is interesting to see how close the characteristics of the measured features are to the characteristics of the ideal features. Plots of the features for different sizes and orientations of each object are shown in figures 2,5 and 12. Each feature was measured for 5 different object sizes and 4 different orientations at each size. The features are plotted against image area.

### 8.2.1 The Feature, $\frac{A}{P^2}$

There are several striking aspects of the plot of $\frac{A}{P^2}$ shown in figure 2. First, there are three objects which have very distinct values of the feature and six which have very similar values. Second, the feature seems to vary significantly for different sizes and orientations of a single object.

It is clear from figure 2 that this feature is quite useful for distinguishing a circle from any of the other objects but not particularly useful for differentiating the pliers from the strippers. In fact, the usefulness of this or any other feature is heavily dependent upon the nature of the objects being recognized. For the objects used in these tests, it was apparent that the $\frac{A}{P^2}$ would have to be used in conjunction with other features to achieve reasonable recognition accuracy.

From the plot, it is also evident that $\frac{A}{P^2}$ is not invariant to either the size or the orientation of the object. Earlier it was shown that this feature would ideally be invariant with respect to both size and rotational variations. There are three primary reasons for these variations. First, the perimeter is a microscopic feature, whereas the area is macroscopic. Second, the digitization process results in images which are not processed in a way that is invariant with respect to object rotation. Finally, quantization errors cause the feature to vary.

The perimeter is a microscopic feature which depends very strongly on the the fine detail of the contour. If a straight segment of a contour is changed slightly so that every over point is offset by one pixel, the length of the line will appear to lengthen by $\sqrt{2}$. Therefore, one can not really expect the perimeter to be unchanged as the object is rotated because the quantization effects will be different for each

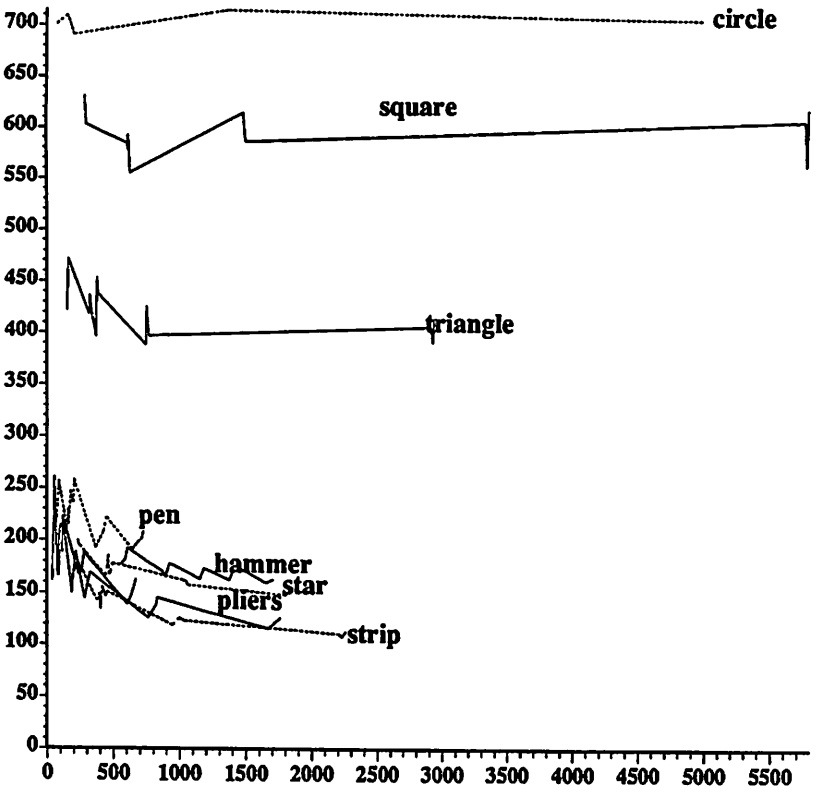



**Figure 2. $\frac{A}{P^2}$ vs. Area for the objects**

rotation and hence the perimeter will vary. This effect can be seen in figure 3, a plot of the area, perimeter and $\frac{A}{P^2}$ for the square for different orientations of the square. The area varies little, while the perimeter has a variation of 7.5%, which results in a variation of about 14% in $\frac{A}{P^2}$.

Other variations in $\frac{A}{P^2}$ can be attributed to the way in which the image is digitized. The camera generates a video signal which consists of two interlaced fields. Every other line of the image is actually in a different field and is therefore separated by one field time (16 *msec*) and not by one line time (75 $\mu Sec$). With this scheme, an $n \times n$ operator becomes a $2n \times n$ operator, because consecutive lines in the video image are really two lines apart in the original image. As a result of this, operators that were intended to symmetrically expand the image (e.g. the logical convolver with a symmetric mask), will actually expand the image more vertically than horizontally. This will distort images that are not radially symmetric as they are rotated. This effect is most pronounced for objects that are very "thin", such as the

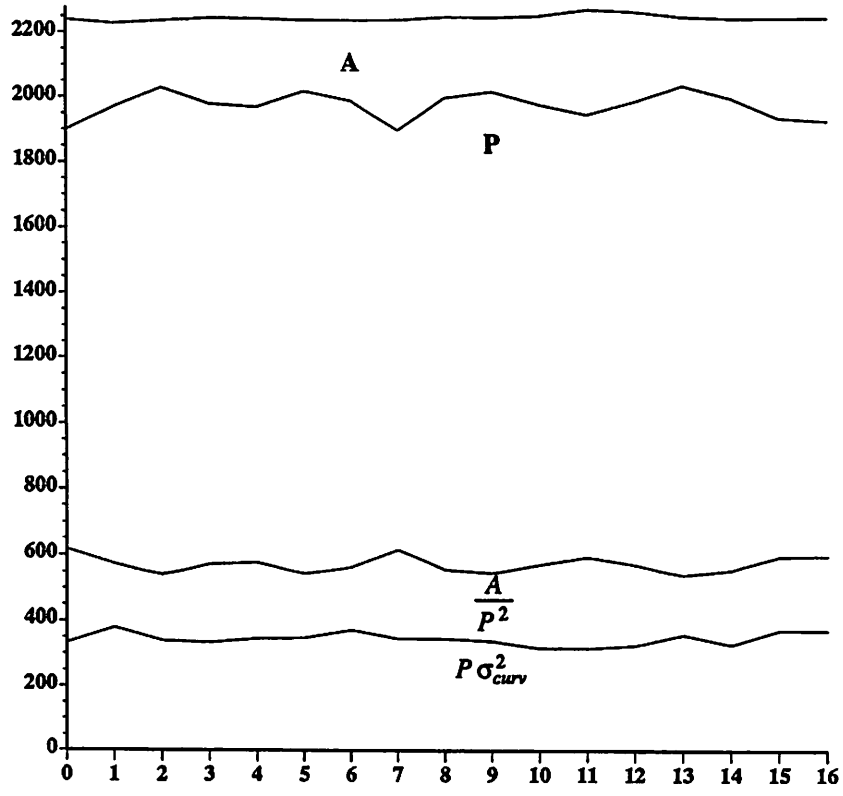



**Figure 3. area, perimeter and $\frac{A}{P^2}$ vs. Orientation for the square**

pen. The area and $\frac{A}{P^2}$ for different orientations of the pen are shown in figure 4. For this object, the area varies by about 20% and accounts for most of the variation in $\frac{A}{P^2}$. For orientations near the center of the plot, the pen is nearly vertical. The area is greatest when the major axis of the pen is horizontal and the greater vertical expansion more strongly affects the area.

Another cause of feature variation is error due to spatial quantization. Earlier it was mentioned that quantization errors affect the perimeter for all object sizes. In addition to this, the area is affected by quantization errors. However, this affect is only severe when there are "thin" parts within an object or the object is small. Therefore, one would expect to see more variation in the feature, $\frac{A}{P^2}$, when the object area is small. In general, much more variance is seen in the curves shown in figure 2 as the area approaches zero.

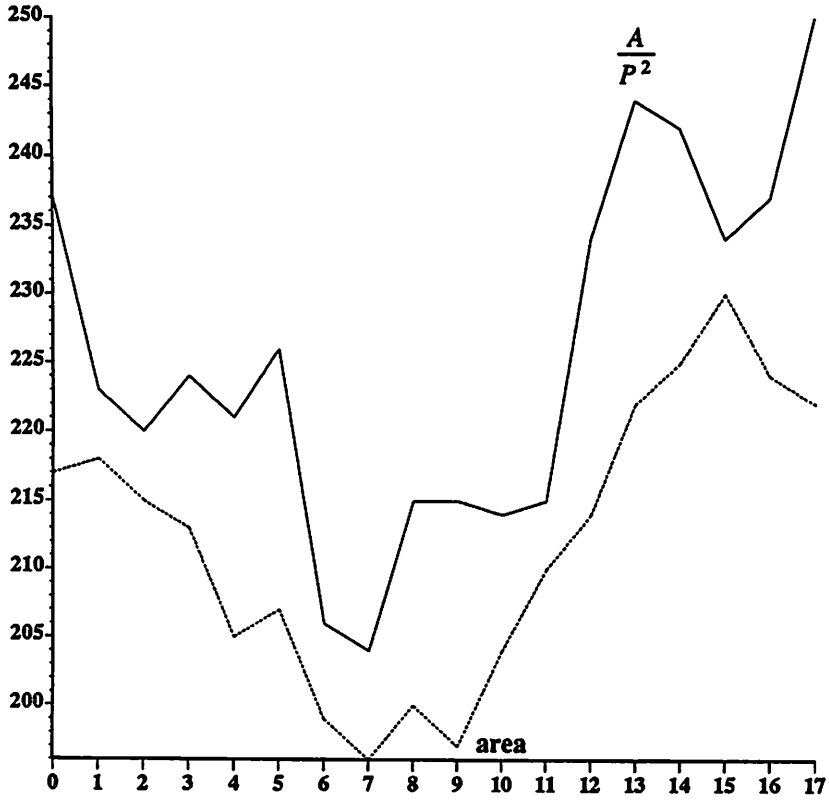



**Figure 4. area and $\frac{A}{P^2}$ vs. Orientation for the pen**

### 8.2.2 The Feature, $P\sigma^2_{curv}$

The feature, $P\sigma^2_{curv}$, for the objects for different sizes and orientations is shown in figure 5. This feature has some very different characterists than $\frac{A}{P^2}$. The most notable characteristic is the very large change in the feature which occurs as the orientation and size of the objects are varied. In addition, the objects that were clustered in the $\frac{A}{P^2}$ space are, in general, spaced farther apart in this feature space.

The ideal analysis showed that this feature should be invariant with respect to object rotation. This is not the case since features for objects with the same size but different orientations can be seen clustered closely together. The quantization errors and non-isotropic image processing seem to be a major cause of these variations. It is also interesting that the curves all tend to slope downward as the area, and the scaling factor $k$ from previous analysis, approaches zero. This was predicted for the case in which the object is dominated by sharp corners and $P\sigma^2_{curv} = a - \frac{b}{P}$ (derived in section 3.4.4).

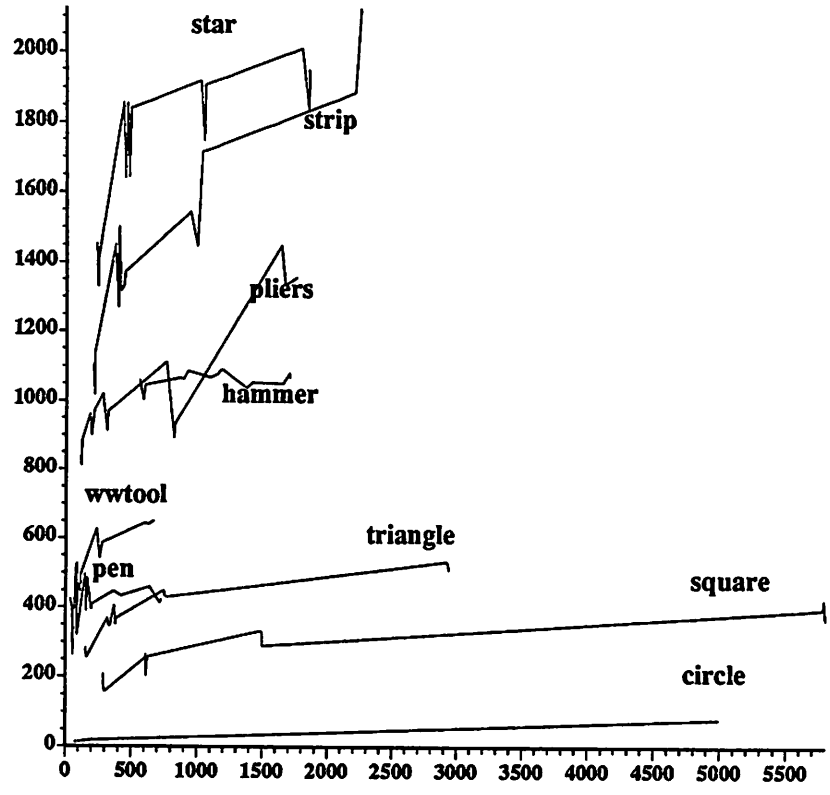



**Figure 5.** $P\,\sigma^2_{curv}$ **vs. Area for the objects**

The feature varies more with respect to size and rotation for the objects with fine detail (e.g. the pliers and the strippers) than for the objects without fine details (e.g. the square and circle). This basically is a problem with under-sampling caused by the down-sampling in the contour tracer and non-isotropic processing. The fine details are lost for some orientations and also as the object is scaled down. This problem is clearly illustrated in curvature plots shown in figures 6 and 7 and figures 8 and 9. In figure 6, some of the fine detail (point of attachment for the spring) of the wire strippers does not show. Figure 7 is the curvature for the strippers (of the same size as that in figure 6 but a different orientation) when the fine detailed was clearly visible. In figure 7, the amplitude of the spikes in the region marked "fine detail" is greater and there are also negative-going spikes. A similar problem occurs when the pliers is rotated. The curvature signals shown in figures 8 and 9 are quite different in the areas marked "jaws". In figure 9, the interior of the jaw was filled in and the large curvature spike evident in figure 8, is not present.

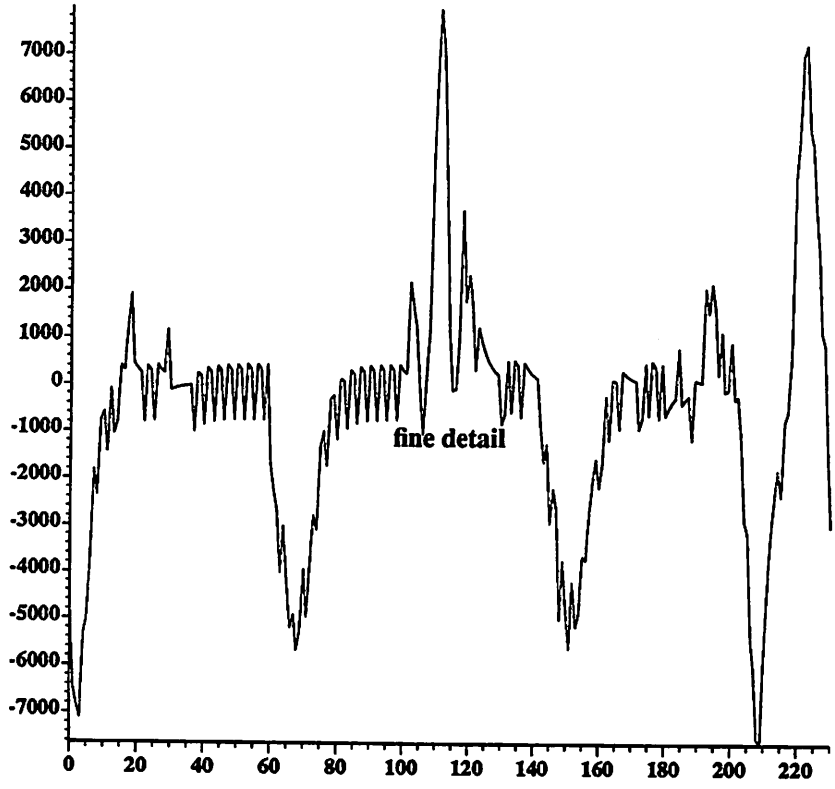



**Figure 6. Filtered Curvature for the Strippers**

Feature variations are also evident from the plots of the features vs. area (figure 10) for a fixed orientation and vs. rotation (figure 11) for a fixed size for both the strippers (solid) and the pliers (dotted). In figure 10, the value of $P\sigma^2_{curv}$ for the pliers is relatively constant until the area reaches about 800. At this point $P\sigma^2_{curv}$ increases because the space between the two jaws becomes more pronounced and a greater curvature spike is produced. The $P\sigma^2_{curv}$ for the strippers is relatively constant. The feature varies more with respect to object rotation. The feature peaks for the strippers near the center of the plot (figure 11) as this is when the fine detail becomes clearly visible. For the pliers, $P\sigma^2_{curv}$ has a local minimum near the center of the plot because the fine detail in the jaws becomes obscured. In both of these cases, part of the problem is due to the non-isotropic expansion. For rotations corresponding to the center of the plot, the jaws are aligned nearly horizontally and the greater expansion in the vertical directions tends to fill in the area between the jaws. This causes some spikes in the curvature signal to disappear. A similar situation exists for the strippers when the fine detail disappears. Because the features vary fairly smoothly as the angle of orientation is varied, non-isotropic processing seems to be the

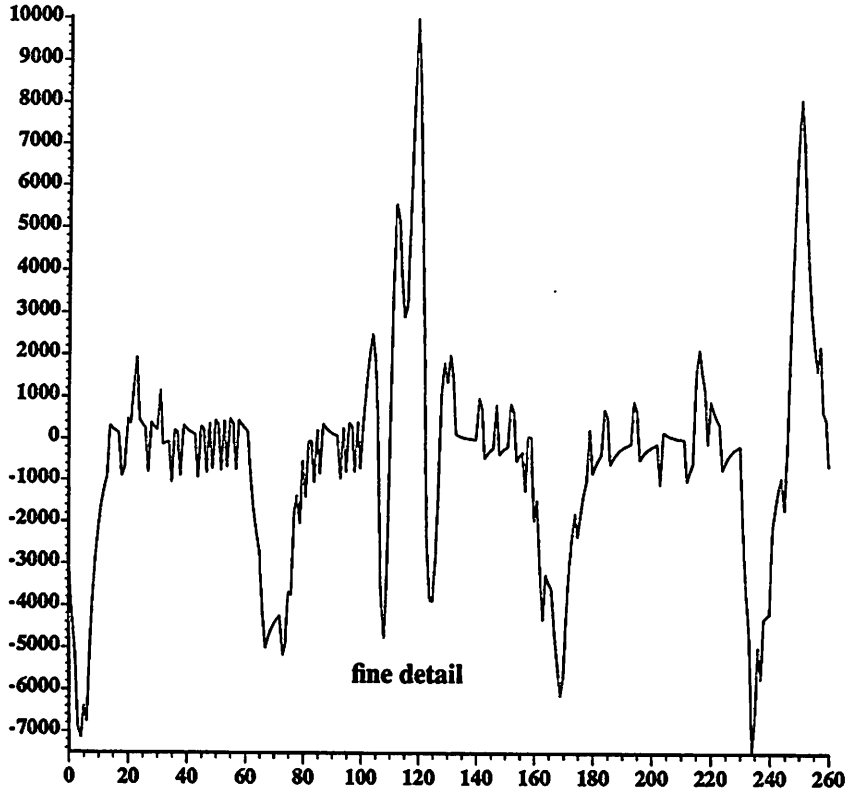



**Figure 7. Filtered Curvature for the Strippers**

dominant cause of feature variation.

### 8.2.3 The Feature, $\sum_{curv>0} curv$

A plot of $\sum_{curv>0} curv$ for the objects for different sizes and rotations is shown in figure 12. Because this feature is computed from the filtered curvature, the problems cited for the $P\sigma^2_{curv}$ in general apply here. At first look, this feature appears to be very sensitive to both the orientation and size of the object.

Because this feature measures the "how convex" (and is ideally zero for all convex objects) an image is, it does little to distinguish the convex objects: pen, square, triangle and circle. In fact, when $\sum_{curv>0} curv$ is below 300, there appears to be little information in the feature. For the convex shapes, the feature value is dominated by quantization noise in the curvature. To reduce the noise, the curvature could be filtered with a lower cutoff filter, but this filtering would also obscure the fine details and cause the type of problems cited earlier.

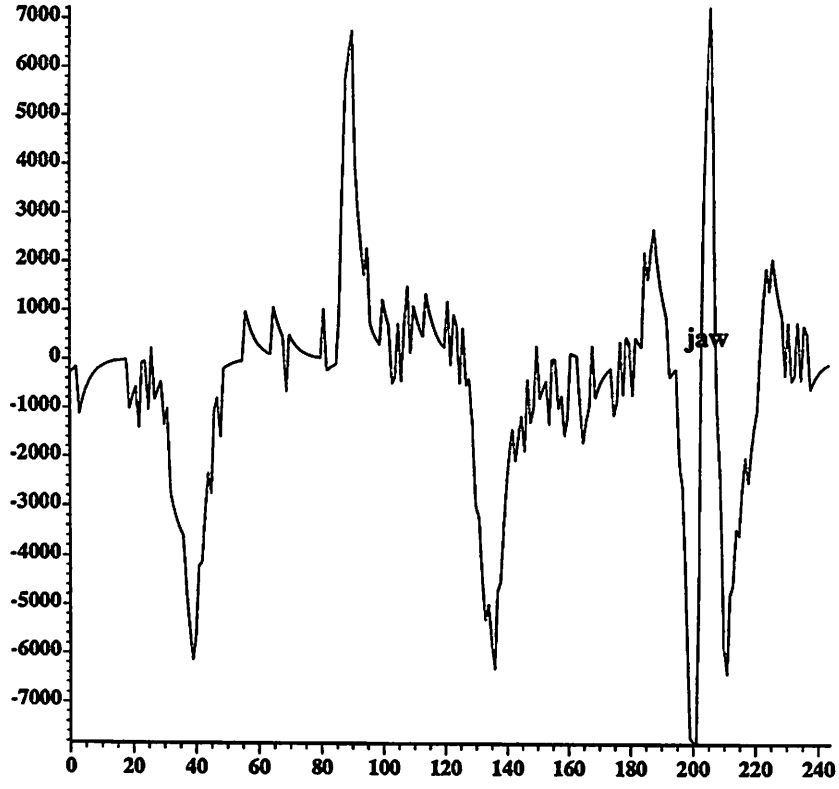



**Figure 8. Filtered Curvature for the Pliers**

### 8.2.4 Use of the Features

From the preceding discussion, it should be clear that the three features used by the recognizer vary significantly with both the size and orientation of the objects. The variation with object size can be accounted for to some extent as discussed in chapter 3 when generating the templates and when recognizing.

The object templates are generated by measuring the features for 4 object sizes and some number of rotations at each size. The area is used as a measure of size because it is in general a more robust feature (the area is a macroscopic feature of the object). An average of the features is obtained for each area and stored along with the area as part of the template. The template features are assumed to be piece-wise linear functions of the area. When recognizing, the area and features of the unknown object are determined. The features for the template objects are found by computing (by linear interpolation) the features that correspond to the unknown object area. A score for each template object is computed

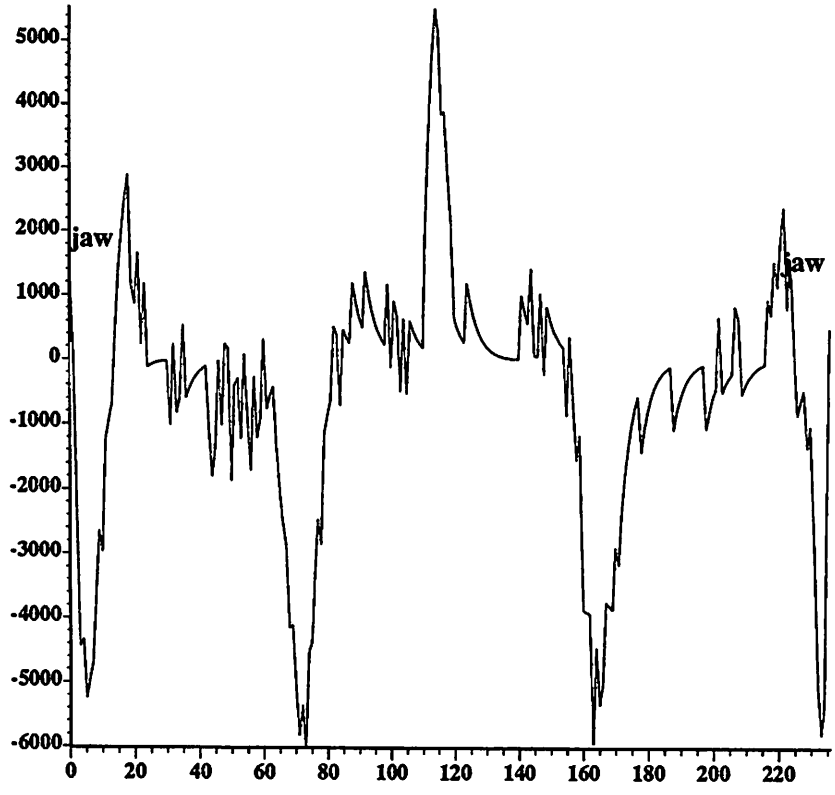



**Figure 9. Filtered Curvature for the Pliers**

according to the equation:

$$score = \sum_{i=feature\ number} W_i \left[ \frac{\overline{f_{i,template}(A)} - f_{i,unknown\ object}}{Min_i + f_{i,unknown\ object}} \right]^2$$

*where* $\overline{f_{i,template}(A)}$ *is the interpolated template feature value*

Essentially, the score for each feature is the square of the percentage deviation of the feature for the unknown object from the feature of the template object. The score for each object is the sum of scores for the features. This basic scheme has some slight modifications. Each feature score is modified by two weights ($W_i$ and $Min_i$). These weights take into account the ability of each feature to distinguish different objects. Ideally, $W_i$ should be a function of $\sigma^2_{feature}$. Objects which have small feature variance should be weighted heavily ($W_i$ large). $Min_i$ is used to reduce the weight of features as the value of the feature decreases. This is used for both $P\,\sigma^2_{curv}$ and $\sum_{curv>0} curv$ which tend to be less reliable as the value of the feature decreases. The features are less reliable for small areas because the features are more sen-

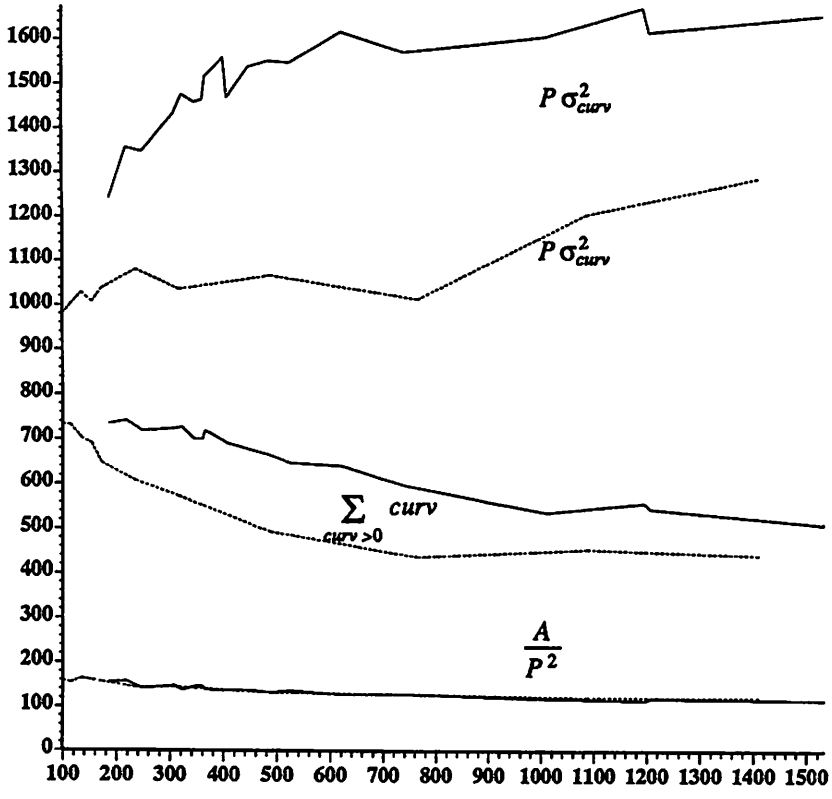



**Figure 10. Features for the Pliers and the Strippers for Fixed Orientation**

sitive to quantization noise or other aspects which are not characteristics of the object.

The weights can be chosen analytically if some assumptions are made and the problem is simplified. Assuming that there is only one feature and there are only two objects, the weights can be chosen by making the probability of mistaking object 1 for object 2 ($P_{err1}$) the same as the probability of mistaking object 2 for object 1 ($P_{err2}$). Assuming that the feature, $X_1$, for object 1 has a Normal distribution (with mean $\mu_1$ and variance $\sigma_1$) and that the feature, $X_2$, for object 2 has a Normal distribution (with mean $\mu_2$ and variance $\sigma_2$), the goal is to choose a threshold, $T$, between $\mu_1$ and $\mu_2$, such that, $P_{err1} = P_{err2}$. If the feature value, $X$, for the unknown object is less than $T$, the object is assumed to be object 1 and if $X > T$, the object is assumed to be object 2. $T$ can be derived under these conditions:

$$P_{err1} = P_{err2}$$

$$\Rightarrow Pr[X_1 > T] = Pr[X_2 < T]$$

$$\Rightarrow 1 - \Phi\left[\frac{T-\mu_1}{\sigma_1}\right] = \Phi\left[\frac{T-\mu_2}{\sigma_2}\right]$$

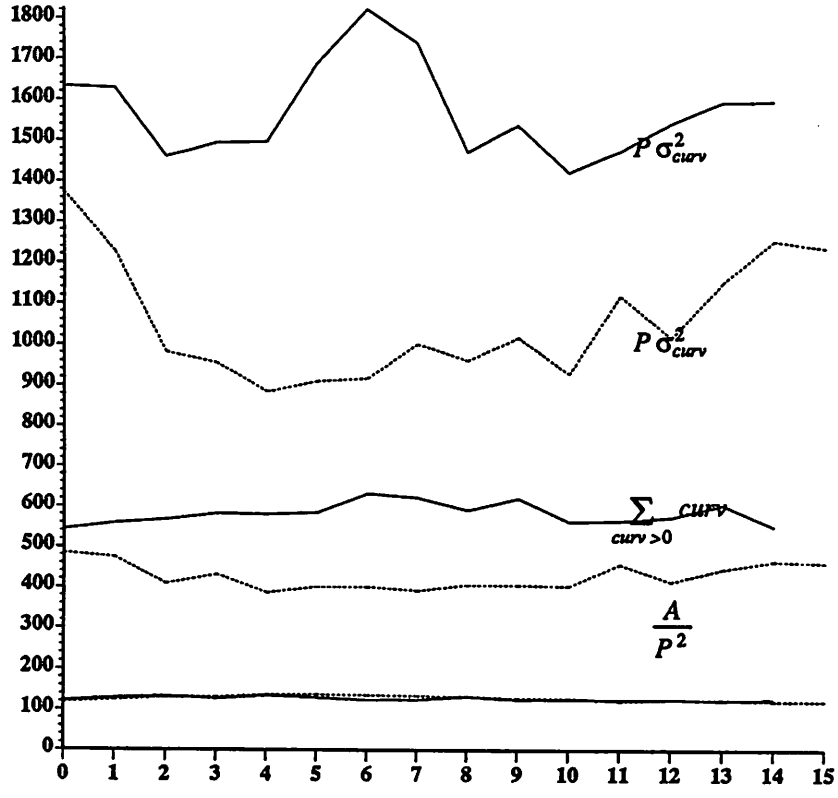



**Figure 11. Features for the Pliers and the Strippers for Fixed Size**

$$let\ T_0 = T - \mu_1$$

$$\Rightarrow 1 - \Phi\left[\frac{T_0}{\sigma_1}\right] = \Phi\left[\frac{T_0 - \Delta\mu}{\sigma_2}\right]$$

$$\Rightarrow \Phi\left[-\frac{T_0}{\sigma_1}\right] = \Phi\left[\frac{T_0 - \Delta\mu}{\sigma_2}\right]$$

$$\Rightarrow -\frac{T_0}{\sigma_1} = \frac{T_0 - \Delta\mu}{\sigma_2}$$

$$\Rightarrow -T_0 = a(T_0 - \Delta\mu) \quad where\ a = \frac{\sigma_1}{\sigma_2}$$

$$\Rightarrow T_0 = \Delta\mu\left[\frac{a}{1+a}\right]$$

$$\Rightarrow T = \mu_1 + \Delta\mu\left[\frac{a}{1+a}\right]$$

$$let\ S_1 = \left[\frac{X - \mu_1}{\sigma_1}\right]^2$$

$$and\ S_2 = \left[\frac{X - \mu_2}{\sigma_2}\right]^2$$

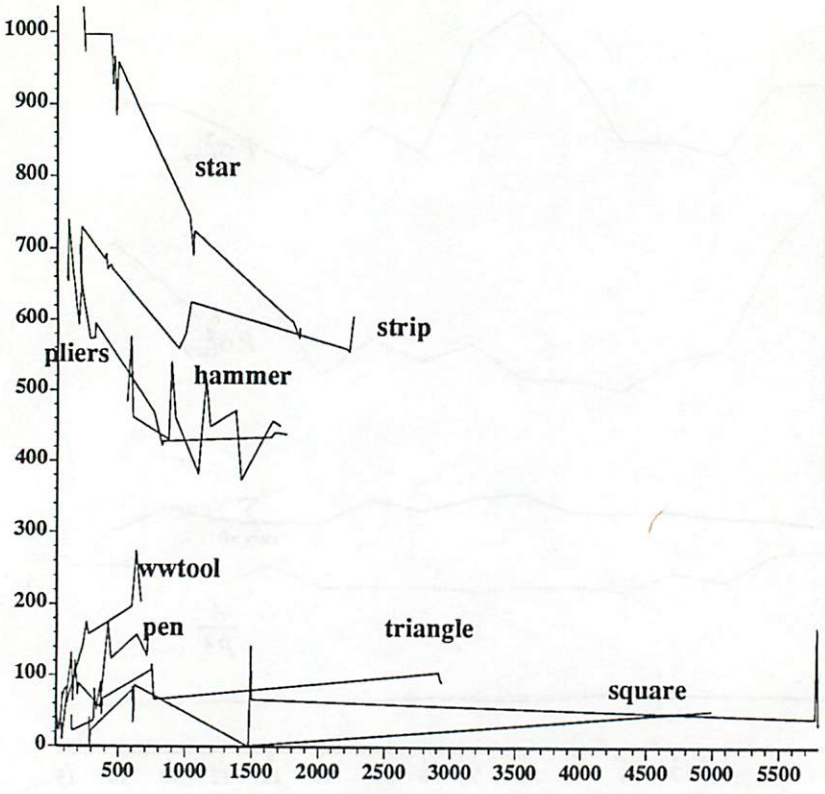



**Figure 12.** $\sum_{curv>0} curv$ **vs. Area for the objects**

If $S_1 < S_2$, the object is assumed to be object 1 and if $S_2 < S_1$, the object is assumed to be object 2. These conditions for making the decision can be shown to be equivalent to those using $T$ directly.

Therefore, under the stated assumptions, the weighting factor is the inverse of the variance of the feature value. The computed weighting function is intuitively satisfying. If the variance of a feature is small, that feature is more reliable and that feature is weighted heavily. However, there is little hope that all of the assumptions that were made are really valid and hence the derived weighting function is not used. The variance of the features is a function of the object size and not constant as assumed above. Instead the variance of the feature values was used to subjectively determine the weights. The values of the weights for the three features are given in table 1. $\frac{A}{P^2}$ is weighted twice that of the other two features because of its lower variance.

| feature | W | Min |
|---|---|---|
| $\frac{A}{P^2}$ | 2 | 0 |
| $P\sigma^2_{curv}$ | 1 | 60 |
| $\sum_{curv>0} curv$ | 1 | 400 |

**Table 1. Feature Weights**

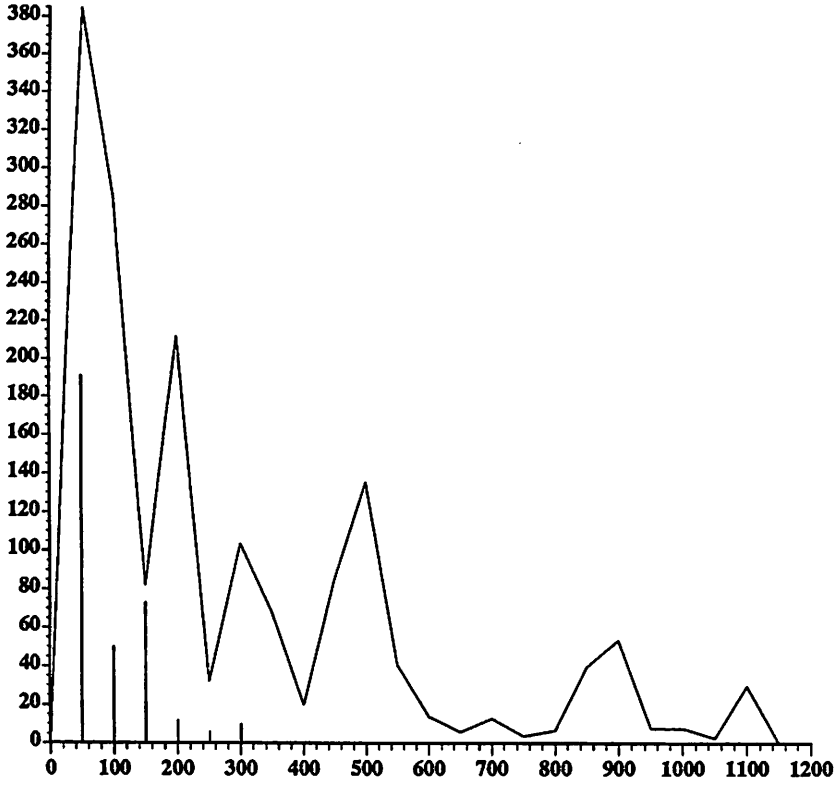



**Figure 13. Recognition Successes and Errors for the pen**

## 8.3 RECOGNIZER OPERATION

The image recognition system can be divided into three main parts, the high rate image processors, the lower rate feature extractors and the pattern matcher and central controller. The high rate processors (sorting filter, linear convolvers, logical convolver, post processors) process all video data in real time and and are not controlled by the central controller. The lower-rate feature extractors (contour tracer and feature extractor) provide the interface between the high rate processors and the slow host computer (pattern matcher and controller). The host computer controller must control the operation of these chips and

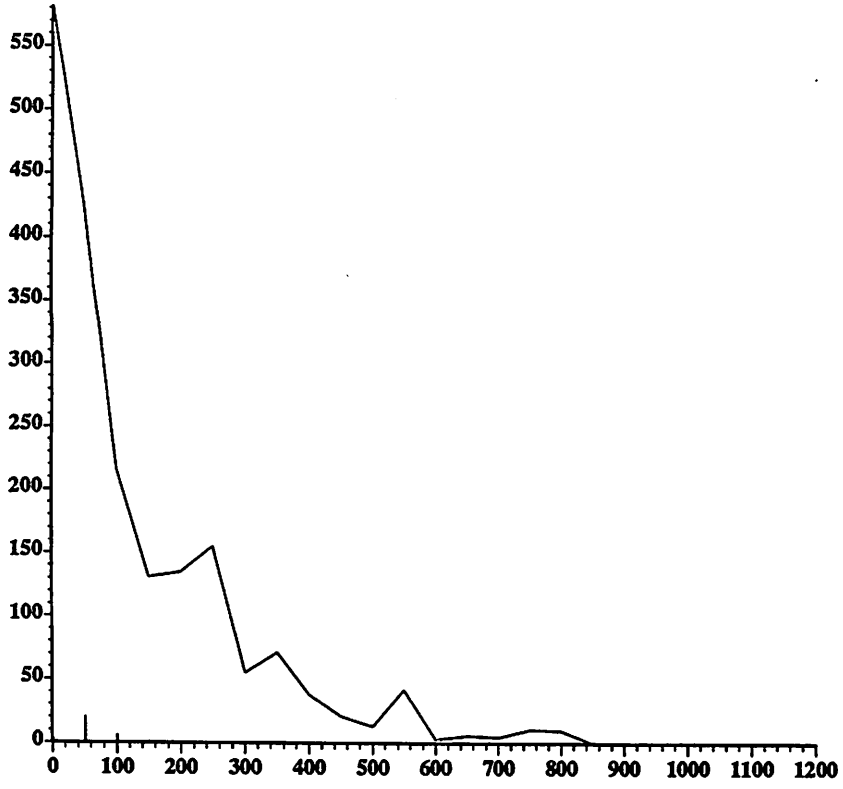



**Figure 14. Recognition Successes and Errors for the wire wrap tool**

read the output results.

The host computer controls the contour tracer operation. It commands the tracer to load a new frame into the internal buffer of the tracer and also restarts the tracing algorithm to try to locate multiple objects in a frame. The host reads the features computed by the feature extractor and the curvature computed by the tracer. After all data for the objects have been read by the host, it computes the $P\,\sigma^2_{curv}$ and $\sum_{curv>0} curv$ features and performs the pattern matching.

The basic tasks that the host performs are diagrammed in figure 17. First, the controller commands the contour tracer to acquire a new frame and search for objects. The same command resets the feature extractor. The controller then waits for the feature extractor to indicate that the tracer has completed its algorithm. If no objects were found in the frame, the process repeats and a new frame is acquired. If the trace ended in an error, the pattern matching is not performed and the controller computes a new start point for the tracer and initiates another trace, still using the old image frame. If the trace finished

successfully, the features are read from the feature extractor and the curvature buffer. The host then computes any additional features and performs the pattern matching. After this, a new start point is computed and the tracer is commanded to begin searching for other objects.

The computation of new start points is one weakness in the current system. Currently, after a successful trace, the tracing is started at the top of the image and at an X position one pixel past the right side of the last object's bounding box. This is a simple way to compute the new start point, but is possible for objects to be masked by larger objects and not be found. If there is an error, the computed right side of the bounding box may be unreliable because the contour of the object touched the right side of the frame. In this case, the previous X start position is increased by a constant and the tracing is performed again. If single point noise is encountered, the bounding box data is useless because the first point is not recognized as valid. The first point is recognized only after the the rest of the contour has been traced and there are at least two points on the contour. For this case, the X start position for the tracer is the X position of the noise plus one.

The controller is also required to detect certain pathological cases. If the tracer starts inside an object, useless data will be generated. Tracing on the inside of an object can be detected by examining the sign of the computed area. If the area is negative, the object was traced from the inside.

## 8.4 RECOGNITION RESULTS

The recognition system was tested with the basic objects shown in figure 1, the same objects used to create the templates. Each object was recognized over a wide range of orientations and sizes. The object rotation was varied by continuously rotating the table the object was on. Size variations were simulated by changing the focal length of the camera lens. A record was kept of the total number of times the recognizer chose each object as a function of the object size.

The recognition results for a system with a median noise rejection filter, no linear to logarithmic conversion, a Sobel edge extractor with pre-determined threshold and single pixel expansion are given in table 2.

All of the errors, except the one for the triangle, consist of mistaking the pen for the wire wrap tool,

| object | recognition | | error |
|---|---|---|---|
| | successes | errors | rate |
| stripper | 1607 | 45 | 2.8 |
| square | 3282 | 0 | 0 |
| circle | 1568 | 0 | 0 |
| star | 2449 | 0 | 0 |
| triangle | 2085 | 1 | .005 |
| pliers | 1755 | 43 | 2.5 |
| pen | 1980 | 342 | 14.7 |
| wwtool | 1946 | 30 | 1.5 |
| overall | 16672 | 461 | 2.7 |

**Table 2. Recognition Results**

the wire wrap tool for the pen, the pliers for the strippers or the strippers for the pliers. This seems to be a reasonable way for the recognizer to fail because the objects that were erroneously picked are actually quite similar (in a subjective sense) to the true object. The number of recognition successes (line) and errors (bar) as a function of object area is plotted in figures 13-16. For the pen and the wire wrap tool, all of the errors are made for relatively small sizes. This is not unexpected because, many of the effects which cause variations in the features are most pronounced for small image sizes. In addition, the pen and wire wrap tool both have the type of shape that causes the value of $\frac{A}{P^2}$ to be sensitive to quantization errors and the non-isotropic processing. It is interesting that the wire strippers were incorrectly identified as the pliers for relatively large object sizes.

The overall error rate for this test was 2.8%. However, if the recognition errors that were made for very small areas (< .6% of the frame area) were ignored, the error rate would drop nearly in half to 1.6%.

## 8.5 MORE CONSTRAINED RECOGNITION

Originally it was desired to recognize two dimensional objects with invariance to the size and orientation of the object without many other constraints put on the problem. However, in some cases it is likely that specific information regarding the environment of the objects or the characteristics of the objects can be used to make the recognizer more reliable. In addition to increasing the reliability of the recognition process, it is likely that the computational requirements would decrease. For example, if size

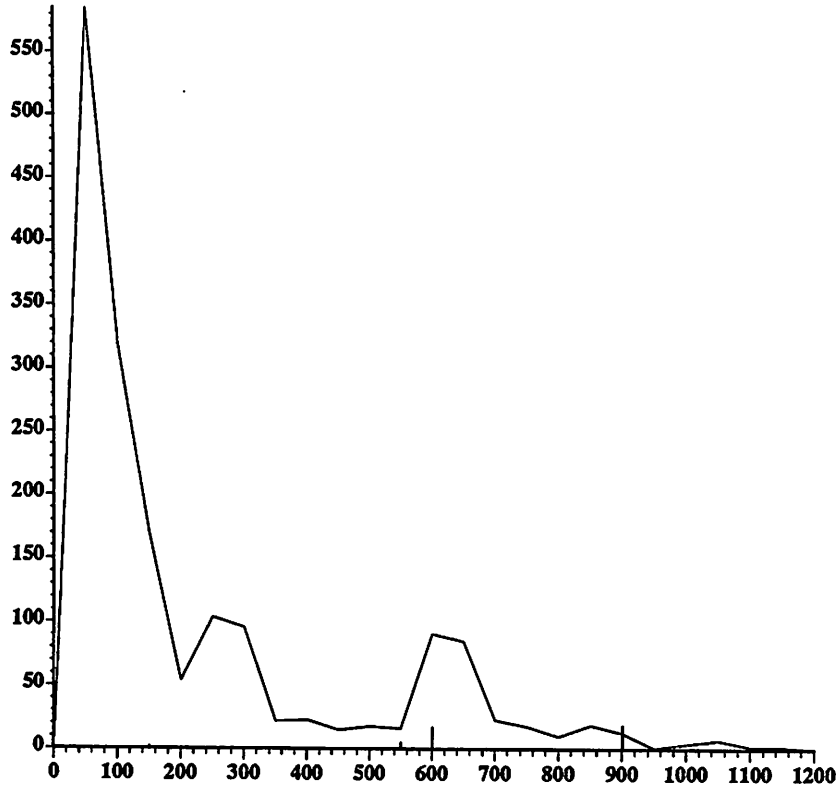



**Figure 15. Recognition Successes and Errors for the pliers**

invariance is not needed, the absolute size of the object will aid the recognition process. If the types of objects to be recognized are known in advance, it may be possible to use features that are better tailored to distinguishing those particular objects.

### 8.5.1 Constant Size Recognition

For some cases it is quite reasonable to assume that the object size will not change. As mentioned in chapter 3, if parts are located on a conveyer belt or other surface in a known plane and the camera is a fixed distance from the known plane, size invariance is not required. In these cases, the absolute size of the image, measured in terms of the area and/or perimeter, can be used to aid the recognition process. In fact, if all objects have different sizes and shapes (as measured by $\frac{A}{P^2}$), then the it should be possible to distinguish the objects solely from the area and perimeter of the object. It is no longer necessary to actually compute $\frac{A}{P^2}$ because size invariance is not needed. In addition, the features do not need to be

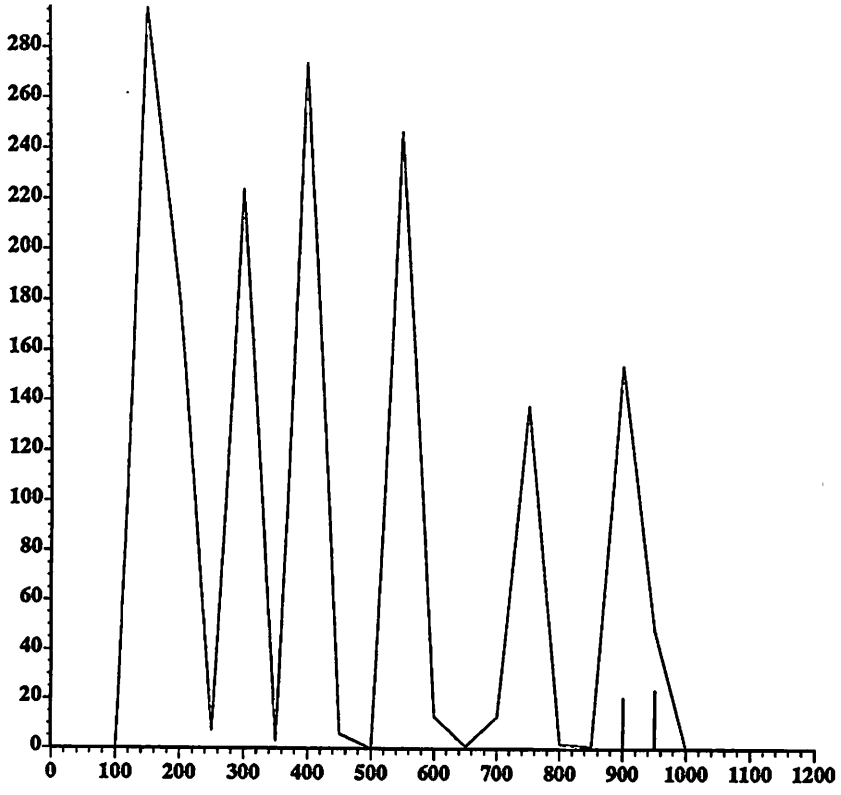



**Figure 16. Recognition Successes and Errors for the strippers**

parameterized with area.

To test the effectiveness of a recognizer using only these two object features, the four objects which were recognized incorrectly in the previous test were tested again without size invariance. The orientation of the objects was varied as in the previous test. The areas of each of four objects along with the recognition results are listed in table 3. All of the objects in this test filled only a small portion of the screen; the largest strippers that can be displayed has an approximate area of 2300 $units^2$ or about 7 times larger than the object used in this test. The problems with variations in area and perimeter were not severe enough to cause recognition errors with these object sizes.

### 8.5.2 Distinct Shape Recognition

An example of another constrained problem is the recognition of objects with distinct shapes as measured by $\frac{A}{P^2}$. If it is desired to recognize the circle, square and triangle, the $\frac{A}{P^2}$ feature seems to be

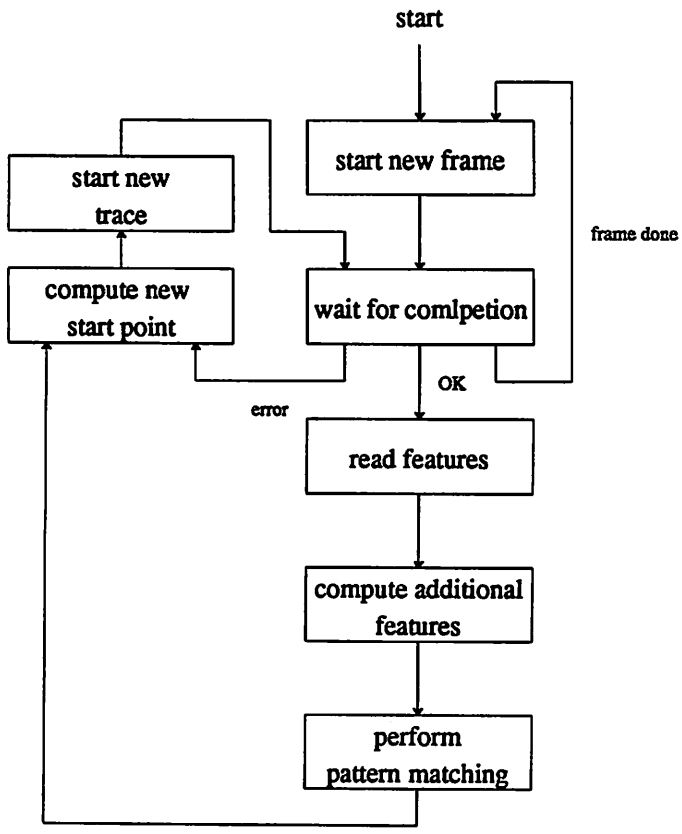



**Figure 17. Recognizer Operation**

| object | area | perimeter | successes | errors |
|---|---|---|---|---|
| pen | 156 | 87 | 3868 | 0 |
| wwtool | 77 | 68 | 2628 | 0 |
| pliers | 178 | 111 | 3317 | 0 |
| strippers | 373 | 168 | 2480 | 0 |

**Table 3. Recognition Results for Constant Size Objects**

adequate to differentiate these objects. This might be used is in very specific problems, such as, having a robot locate a ping-pong ball and then hit or catch it.[1]

The recognizer was tested using only the $\frac{A}{P^2}$ feature to recognize the circle, square and triangle over a wide range of orientations and sizes. Although no attempt was made to recognize the other objects, they were still template objects and hence the recognizer could still mistakenly choose one of

[1] This has been suggested as a goal to demonstrate the abilities of robots in the control systems group at U.C.B.

them. The results of this test are shown in table 4.

| object | successes | errors |
|---|---|---|
| square | 1756 | 0 |
| triangle | 2116 | 0 |
| circle | 1494 | 0 |

**Table 4. Recognition Results for Distinct Shape Recognition**

It should not be surprising that the recognizer made no mistakes in this test because the values of $\frac{A}{P^2}$ for these objects (figure 2) are quite distinct. This test demonstrates the ability of the recognizer to achieve good recognition accuracy for specific objects using a feature which can be computed easily.

### 8.5.3 Using Color

Until this point there has never been any mention of the use of color information to aid in either edge detection or recognition. To rely only on the intensity of the object to find edges is a handicap unless all objects of interest are black and white. Changes in luminance as well as hue should be used to find edges. In addition, hue information could simplify some recognition problems. For example, the wire strippers and the pliers have different colored handles (one red, one blue). It would be possible to have the image processor count the number of pixels with each color within the boundary of the objects after contour tracing. If the information from the other features was not sufficient to make a reliable decision (i.e. the scores for the two objects were close), the color information could be used.

### 8.5.4 Recognition without Edge Extraction

The problems with thresholding the gray-level image were discussed in chapter 2. In environments with good contrast and low noise, it may be possible to simply threshold the gray-level image and use this as an input to the contour tracer. This was in fact done when the contour tracing algorithm was being simulated because the convolver chips were not yet available. If the gray-level image is thresholded directly, not only is the edge extractor and logical convolver unnecessary, but the 8-bit A/D converter could be replaced with a comparator.

### 8.5.5 Achieving Size Invariance Optically

There is no reason why the invariance with respect to object size of the recognizer must be achieved algorithmically. It could instead, be obtained optically, by changing the focal length of the lens and perhaps translating the object or camera in order to have the object fill the frame. If this were done, features would not have to be used that are reasonably invariant with respect to object size. In addition, the recognizer would not have to deal with small images which are more sensitive to quantization errors. If the object were scaled to fill the frame, quantization errors would be as small as possible. Higher resolution may be more economically obtained through this method compared to simply sampling the object at a higher rate. The value of this scheme can be seen by noticing that this kind of size normalization would make the recognition problem the same as that in section 8.5.1. By making the objects a standard size, no recognition errors were made when using only two features which were quite easy to compute.

### 8.5.5 Conclusions

We tried to solve a relatively general recognition problem and hence used some general techniques and features which resulted in recognition errors. For a specific recognition problem, as much specific information regarding the objects and the environment should be utilized to make the recognizer as robust as possible and to also reduce the computational requirements of the system. It has been shown that specific recognition problems can be solved with fewer computations than the general problems and with a greater accuracy.

## 8.6 IMPROVING THE RECOGNITION ACCURACY

The basic causes of feature variations and hence recognition errors, include quantization errors in the spatial image and the curvature signal, non-isotropic image expansion and the use of the perimeter which is a microscopic property of the object.

Both spatial and curvature quantization errors can be reduced by simply increasing the resolution of the system. The tracer down-samples the image to 128 x 128 pixels. With a more advanced technology, a larger buffer could be put on-chip and the down-sampling would no longer be necessary. If no down-sampling were performed a significant increase in resolution would result and hence quantization

errors would be minimized. Although the curvature is inherently quantized to three bits, the effects of quantization errors can be minimized by over-sampling the curvature (which happens as the spatial resolution is increased) and then removing quantization noise by low-pass filtering.

As mentioned above, it may be more economical to achieve greater resolution by scaling the image before digitizing to efficiently use the available resolution.

The problems associated with under sampling are made worse by the logical convolver. Although expansion of the binary image behaves like a low pass filter in some sense (i.e. image details are blurred), it can actually create high spatial frequency components in the image by shrinking the background. For this case, it is possible that aliasing will occur when the image is down-sampled if the background region becomes narrower than the sampling interval.

The non-isotropic processing can be corrected by employing an image digitizer that samples the image on a square array and without interlace. Another way to solve the non-isotropic processing problem is to modify the image processing chips to take into account the particular distances between pixels horizontally and vertically. For example, to compensate for greater expansion vertically, the "impulse responses" could be lengthened in the horizontal dimension. It may be advantageous to actually digitize the image on a hexagonal grid, where each pixel is equidistant from each of its six nearest neighbors.

The perimeter may have to be discarded as a feature as there is no obvious way to reduce the variance of the perimeter as the object is rotated.

## 8.7 OTHER RECOGNITION TECHNIQUES

Two other recognition techniques were examined. The performance of these techniques was compared to the performance of the feature-based recognizer. Both the Fourier descriptors and curvature matching techniques were characterized.

### 8.7.1 Fourier Descriptors

The first eight normalized Fourier coefficient magnitudes were used as features to recognize the objects in figure 1. The same basic system configuration was used that was used in the feature-based recognizer. In this case, the curvature values were used to reconstruct the (X,Y) pairs which are needed

to compute the Fourier coefficients. Because the Fourier coefficients were computed on the host, the total number of recognition tests was reduced. The images which were used to generate the templates for the feature based recognizer were used as test images (about 20 images/object). The system was trained on one large image per object and no attempt was made to allow for size variance as discussed previously. The error measure used was a simple Euclidean distance between the template and unknown coefficient vectors.

| object | attempts | errors | error object |
|---|---|---|---|
| star | 20 | 0 | - |
| square | 20 | 0 | - |
| triangle | 20 | 3 | circle |
| pen | 20 | 1 | wwtool |
| wwtool | 20 | 1 | pen |
| pliers | 20 | 2 | triangle |
| strippers | 20 | 0 | - |
| circle | 15 | 0 | - |
| total | 155 | 7 | |

**Table 5. Recognition Results for Fourier Descriptors Features**

The results of the recognition are given in table 5. There are some similarities between these results and those for the feature based recognizer. The pen and wire wrap tool were mistaken for each other. In addition, the objects which were recognized correctly most of the time in the feature based recognizer were also typically recognized correctly in this test. However, the pliers was mistakenly called the triangle and the triangle was called the circle.

| coeff | $\sigma^2$ |
|---|---|
| $F_1$ | .42 |
| $F_2$ | .36 |
| $\sqrt{F_1^2+F_2^2}$ | .12 |

**Table 6. Variance of Fourier Coefficients for the Triangle**

There are two problems with this technique. The first is that by computing Fourier coefficients, it is being assumed that the contour is uniformly sampled. However, for a square sampling grid, the diagonal points are farther apart than the points that are aligned vertically or horizontally. This could be

corrected by sampling the image on an hexagonal grid. The other problem is that the coefficients tend to be somewhat correlated. That is, if one coefficient is large, another may tend to be small. For example, for the triangle, the first and second normalized coefficients (the two largest ones) have a correlation coefficient of -.85. In table 6 it can be seen that the variance of the two coefficients is larger than that for the square root of the sum of the squares of the coefficients. For some cases the first coefficient is large and the second is small. In other cases, the converse is true. The exact magnitude of each coefficient is probably determined by quantization errors and the non-uniform sampling. This results in recognition errors in a system in which this relationship is ignored.

| object | attempts | errors | error object |
|---|---|---|---|
| star | 20 | 0 | - |
| square | 20 | 1 | circle |
| triangle | 20 | 2 | pen, wwtool |
| pen | 20 | 4 | wwtool |
| wwtool | 20 | 8 | pen |
| pliers | 20 | 0 | - |
| strippers | 20 | 0 | - |
| circle | 15 | 0 | - |
| total | 155 | 15 | |

**Table 7. Recognition Results when matching Curvatures**

### 8.7.2 Curvature Matching

A recognizer which matches curvatures was tested in a similar way to the Fourier descriptors recognizer. A curvature template for each object was generated from a single image of an object which fills the frame. The images which spanned a wide range of orientations and sizes were then used to test the recognizer.

Size invariance is obtained by simply making the template curvature and the curvature of the unknown object the same length. This is done by down-sampling the longer curvature. Because the curvature is the second difference of the position of the contour, it has significant high frequency content and can not be simply down-sampled without aliasing. To prevent these problems, when a point is removed from the curvature, the value of that point is added to an adjacent point. In this way the slope, which is

the discrete integral of the curvature, is not changed significantly. Because sharp corners can be represented by an impulse in the curvature signal, it is important not to simply discard points or major object features may be lost. This simple matching technique requires that the relative distance between corners be maintained as the image is rotated and its size is varied. Therefore, a correction must be made when there is a diagonal step in the contour, because diagonal points are actually $\sqrt{2}$ units, not 1 unit apart. To accomplish this, one dummy zero point is inserted into the curvature after two diagonal points are encountered on the contour. This makes the diagonal points appear to be approximately 50% farther apart than the horizontal or vertical points.

Rotational invariance is achieved by shifting the template relative to the curvature of the unknown object and using the best match over all shifts. The two curvature signals are filtered by the same filters used in the feature-based recognizer. The error between the scaled curvatures is found for each shift by accumulating the square of the error between the two at each point.

The results of the recognition process are summarized in table 7. Again, mistakes were made between the pen and the wire wrap tool. It is interesting that the pliers and strippers were not confused with each other. However, the square was mistakenly called the circle and the triangle was called the pen and the wire wrap tool. Considering that only a single template curvature was stored for each image, the recognizer performed reasonably well.

The use of the curvature in this way is fairly sensitive to errors in quantization of the object. If quantization errors make a normally smooth side of an object become jagged, its apparent length, measured by the normalized number of pixels along the edge, will increase. This could cause the object to appear different from itself when the quantization errors cause the side to appear shorter. An example of this is a triangle with one side aligned with one axis of the digitizing grid and another side that is not vertical, horizontal or diagonal. The side aligned with the digitizer axis will have zero curvature because the pixels are perfectly in line. The off-axis side will have non zero curvature because the particular slope of the line is not one of the 8 which can be exactly represented. As a result the slope will change between the two values closest to the true slope of the line. The line will appear "jagged" and hence will appear longer relative to the edge aligned with an axis. The net result of this is that the shape becomes distorted,

with errors in the relative distances between sharp corners. Basically, this parameterization of the curvature has the same problems of the perimeter because the perimeter is simply the integral of the curvature parameter.

One solution to the problem is to find the true X,Y position of points on the contour which have large curvature values. The points with large curvature are the ones which can contribute large errors in the matching and hence should be positioned accurately. The distance between points of large curvature would be known within the spatial resolution of the system. The curvature could then be non-linearly scaled to ensure that these points are the proper relative distance apart.

## 8.8 CONCLUSIONS

Although none of these recognizers produced perfect results, they do show that high-speed object recognition can be performed with a very simple custom system. By utilizing custom chips to perform the bulk of the high rate processing and mid-rate feature extraction, the entire system could be made to run at nearly real-time rates. With minor improvement to the system, real-time operation is likely to be obtained. Further, it was found that a very simple feature-based recognizer performed comparably to much more computationally intensive recognizers. The performance of the feature-based recognizer could be improved by using greater resolution, sampling the image on a square grid and/or using optical size normalization.

The use of hardware in an hierarchical manner and the appropriate CAD tools made it possible to develop a working set of image processing chip and the recognition system in only 1.5 man years. By making tradeoffs between the algorithms, architectures, circuits, and CAD tools, large savings in the total design effort could be achieved.

# APPENDIX A. THE RECOGNITION BOARD

The recognition system described in chapters 3 and 8 was built on a board. The way in which the custom chips are connected on the recognition board is shown in figure 1. The connections are fairly straight forward for the chips that produce video outputs (they are connected in a simple cascade fashion). There is a much more complicated interconnection between the contour tracer and feature extractor. In addition to the connections shown in figure 1, there are circuits to latch the control words from the host, send results from the board to the host, generate the control signals for the contour tracer and the feature extractor and generate the clocks for the custom chips.

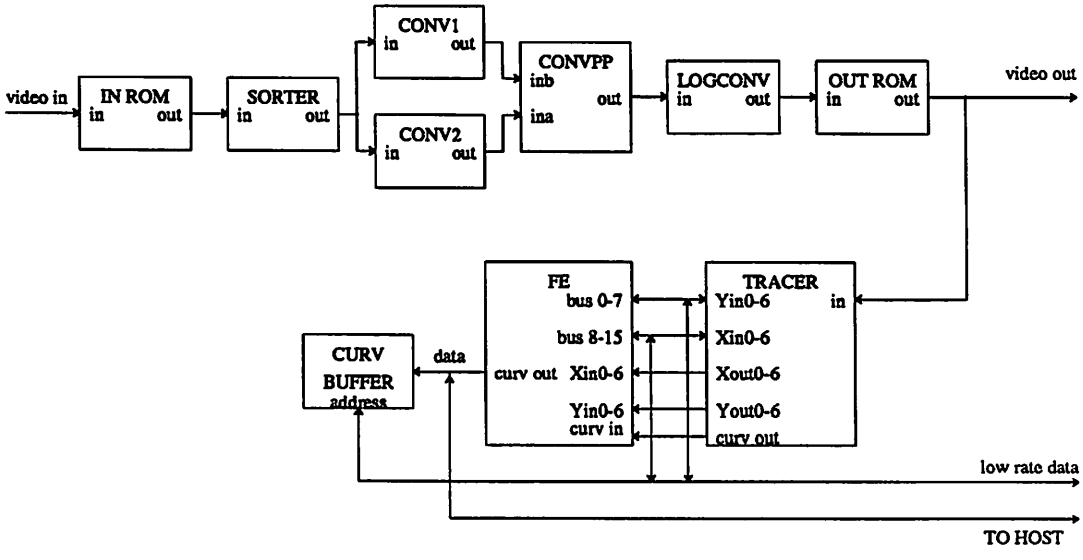



**Figure 1. Custom Chip Interconnection**

## A.1 DESCRIPTION OF CHIP SIGNALS

The pinouts for the various chips are given in tables 3-10. For each chip a brief description of all signals is given below. For all signals that are numbered (e.g. din0-7) the lsb is the lowest numbered signal (e.g. din0) and the msb is the highest.

LUT ROMs

| | |
|---|---|
| din0-7 | video input |
| dout0-7 | video output |
| enable | modify video when high, pass data when low |

Logical Convolver

| | |
|---|---|
| din0-7 | video input |
| dout0-7 | video output |
| bypass | modify video when low, pass data when high |
| bloat0-3 | impulse response select - to bloat by n set only bloat0-bloatn high |
| start | start of video line synchronization |

Convolver Post Processor

| | |
|---|---|
| a0-7 | video input |
| b0-7 | video input |
| dout0-7 | video output |
| g1,g2 | select gain for input b<br>g1=0, g2=0, gain=1<br>g1=1, g2=0, gain=2<br>g1=0, g2=1, gain=4<br>g1=1, g2=1, gain=8 |
| t3-t7 | offset (or threshold) value, LSBs are zero |
| negclip | when high map positive gray values to entire range<br>when low map positive gray values to top half of the gray values and negative gray values to lower half |
| binarycon | set all bits but the MSB to zero when high (for thresholding) |
| zeroa | zero video input a |
| absval | modify both a and b inputs by full wave rectifier |

3x3 Convolver (ROM based coefficients)

| | |
|---|---|
| din0-7 | video input |
| dout0-7 | video output |
| sel0-3 | impule response select |
| start | start of video line synchronization |

3x3 Convolver (programmable coefficients)

| | |
|---|---|
| din0-7 | video input |
| dout0-7 | video output |
| con0-7 | control (coefficients) signal inputs (see table below)<br>for MAC:<br>only one of sh0*-sh4* should be low (selects coeff magnitude)<br>invert high gives negative coefficient<br>zero* low gives zero coeff<br>for GAIN<br>zero high gives gain of 1<br>shn high (zero low) gives gain= $1-2^{-n}$ |
| a0-a3 | register address (see table below) |
| W* | active low write strobe<br>latches con0-7 into register determined by a0-3 |
| start | start of video line synchronization |

programmable convolver register use

| register address | coeff | type | done |
|---|---|---|---|
| 0 | $h_{31}$ | MAC | 0 |
| 1 | $h_{32}$ | MAC | 1 |
| 2 | $h_{33}$ | MAC | 0 |
| 3 | $h_{23}$ | MAC | 0 |
| 4 | $h_{21}$ | MAC | 0 |
| 5 | $h_{22}$ | MAC | 1 |
| 6 | *Gain* | GAIN | x |
| 7 | $h_{12}$ | MAC | 1 |
| 8 | $h_{13}$ | MAC | 0 |
| 9 | $h_{11}$ | MAC | 0 |

Programmable convolver control words

| bit | TYPE | |
|---|---|---|
| | MAC | GAIN |
| con0 | sh0* | zero |
| con1 | sh1* | sh4 |
| con2 | sh2* | sh3 |
| con3 | sh3* | sh2 |
| con4 | sh4* | sh1 |
| con5 | invert | |
| con6 | zero* | |
| con7 | done | |

3x3 Sorter

| | |
|---|---|
| din0-7 | video input |
| dout0-7 | video output |
| c0-c1 | vertical region of operation select |
| c2-c3 | horizontal region of operation select |
| invert* | invert minimum when low |
| zeromin | zero minimum |
| zeromax* | zero maximum or median |
| selmax* | compute maximum when low, median when high |
| start | start of video line synchronization |

| function | invert* | zeromin | zeromax* | selmax* |
|---|---|---|---|---|
| minimum | 1 | 0 | 0 | x |
| maximum | x | 1 | 1 | 0 |
| median | x | 1 | 1 | 1 |
| max diff | 0 | 0 | 1 | 0 |

Contour Tracer

| | |
|---|---|
| yin1-yin6 | Y start value (LSB is always high), normally connected to bus2-7 |
| xin1-xin6 | X start value (LSB is always high), normally connected to bus10-15 |
| ramout | RAM output for testing |
| xout0-7 | Current X position, normally connected to FE xin0-7 |
| yout0-7 | Current Y position, normally connected to FE yin0-7 |
| SRout | input shift register output for testing |
| videoin | video input signal (1 bit) |
| curvout1-3 | output curvature signal, normally connected to FE curvin1-3<br>This output is the actual curvature offset by 4. So when curvout=0 the actual curvature is -4. |
| stateout0-5 | Current tracer state, normally connected to FE statein0-5 |
| error | indicates trace error when high |
| phi1w, phi2w | video clocks for loading data into tracer |
| hsync | write clock during horizontal blank |
| acquire | when high, loads frame into buffer and starts trace |
| load | loads tracer registers for tracing same frame at different location |
| phi1,phi2 | clocks used during trace mode (can be different than phi1w,phi2w) |
| statein0-5 | start state for tracer after "acquire" or "load" (normally 000011) |
| auxram | FSM input for testing |
| selauxram | when high FSM takes input from auxram instead of real RAM output for testing |
| start | start of video line synchronization |
| slow phi1 | slow clock used by tracer, needed to synchronize FE |
| Coyout | carry out from Y counter, used to detect trace off edge |
| Coxout | carry out from x counter, used to detect trace off edge and frame done |
| dx=zero* | indicates that X position is the same as first point on contour |
| dy=zero* | indicates that Y position is the same as first point on contour<br>These 2 signals are used to determine when the contour has been traced |

Feature Extractor

| | |
|---|---|
| statein0-5 | Current tracer state, normally connected to tracer stateout0-5 |
| yin0-yin6 | Y current value, normally connected to tracer yout0-6 |
| xin0-xin6 | X current value, normally connected to tracer xout0-6 |
| curvin1-3 | input curvature signal, normally connected to tracer curvout1-3 |
| curvout1-3 | output curvature signal to buffer |
| bus0-15 | I/O bus for sending data to host, reading data from host, setting buffer address for read/writing and setting tracer X,Y start values |
| latch | signal that causes FE latches to load data from the bus |
| dir | indicates direction of data on bus, used to control tri-state buffers |
| validpoint | strobe to latch curvature data into buffer |
| override | force register to be selected from a2-a0 |
| a2-a0 | register address (address 7 for loading data) |
| reset | resets FE after tracer acquire |
| Coxin | Coxout from tracer |
| dx=dy=0 | indicates that current X,Y position is same as first point on contour |
| error1* | signal to indicate trace errors (computed from tracer signals) |
| load | tracer load signal |
| done* | indicates when the trace has finished |
| outcode0*-1* | code indicating tracer state |

## A.2 BOARD LAYOUT

The layout of chips on the board is shown in figure 2. The host computer and video connectors are at the left and the custom chips are clearly marked.

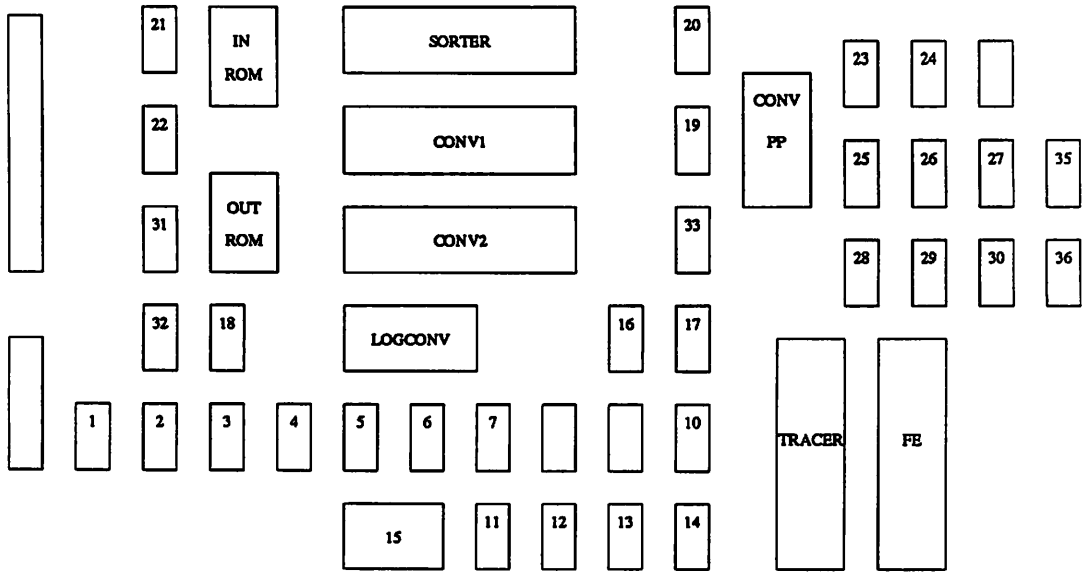



**Figure 2. Board Layout**

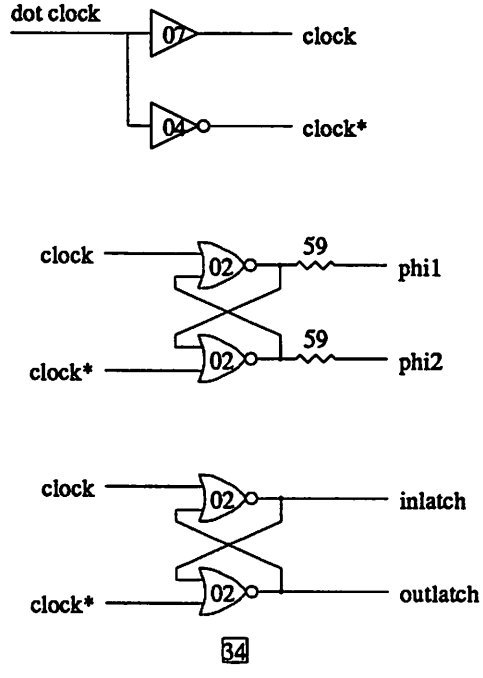



**Figure 3. Clock Generator Circuits**

### A.2.1 The Clock Generators

The clocks for all of the chips on the board are derived from a single phase clock (dot clock) sup-

plied by the video digitizing circuit. All chips were designed to utilize clocks with zero clock separation to prevent the need to generate difficult to control clock separation and to give the circuits as much of the complete cycle as possible for operation. The circuit used to generate the clocks is shown in figure 3. The cross coupled NOR gates generate non-overlapping clocks from clock and clock* (clock is buffered to equalize the delay in clock and clock*). The resistors are utilized to reduce ringing. To get clocks which rise quickly to 5 Volts, 74S02 parts were used with a 7 V supply.

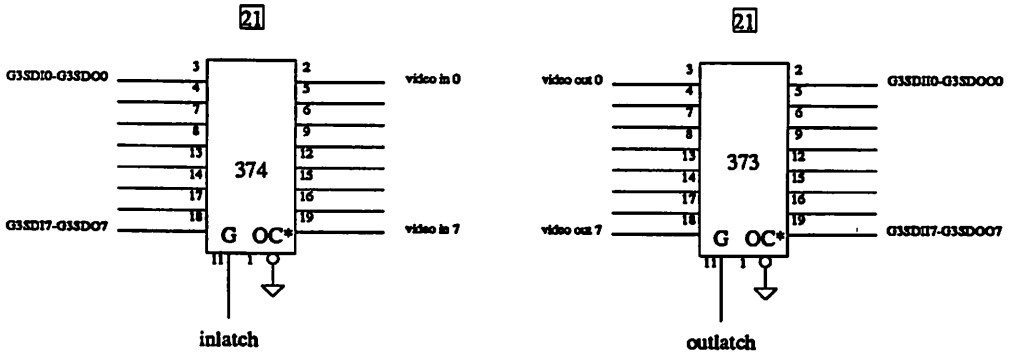



**Figure 4. Video Input and Output Latches**

### A.2.2 Video Interface

The video in (input video from the digitizer) and video out (output video to the display) signals are latched with the circuit shown in figure 4. The circuits are the interface between the video signals on the recognition board and the video signals on the digitizer board which operate off different clocks. Video in goes to the input ROM and video out comes from the output ROM.

All low rate control signals from the host are latched by the circuit in figure 5. This is just a bank of registers that latches data on the Multibus when the appropriate address is decoded. Low rate data is read from the board with the circuit in figure 6. This circuit enables data onto the Multibus when the appropriate address is decoded. The "wsel/" and "sel*" signals are decoded selects from the Multibus. The "BDATn/" signals are the Multibus data signals. The input and output port assignments are shown in table 1 and table 2.

| bit | port | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | sorter zeromax | conv1 impsel0 | convpp thresh3 | conv2 impsel0 | FE a0 | bus0 | bus8 |
| 1 | sorter zeromin | conv1 impsel1 | convpp thresh4 | conv2 impsel1 | FE a1 | bus1 | bus9 |
| 2 | sorter selmax* | conv1 impsel2 | convpp thresh5 | conv2 impsel2 | FE a2 | bus2 | bus10 |
| 3 | sorter c0 | conv1 impsel3 | convpp thresh6 | conv2 impsel3 | FE override | bus3 | bus11 |
| 4 | sorter c1 | convpp negclip | convpp thresh7 | logconv bypass | FE latch | bus4 | bus12 |
| 5 | sorter c2 | convpp absval | convpp binarycon | logconv bloat1 | acquire | bus5 | bus13 |
| 6 | sorter c3 | convpp g1 | convpp zeroa | logconv bloat2 | load | bus6 | bus14 |
| 7 | out ROM enable | convpp g2 | in ROM enable | logconv bloat3 | tracer selauxram | bus7 | bus15 |

**Table 1. Output Port Assignments**

| bit | port | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | FE outcode0* | bus0 | bus8 | tracer xout0 |
| 1 | FE outcode1* | bus1 | bus9 | tracer xout1 |
| 2 | FE done* | bus2 | bus10 | tracer xout2 |
| 3 | tracer error | bus3 | bus11 | tracer xout3 |
| 4 | FE curvout1 | bus4 | bus12 | tracer xout4 |
| 5 | FE curvout2 | bus5 | bus12 | tracer xout5 |
| 6 | FE curvout3 | bus6 | bus14 | tracer xout6 |
| 7 | done | bus7 | bus15 | tracer ram out |

**Table 2. Input Port Assignments**

### A.2.3 Control Signals

Figure 7 shows the circuit that generates the "start" signal for chips with line delays. This circuit detects edges in the horizontal blank signal and generates a single pulse at the start of each line. The contour tracer could use this signal directly if the pixels were on a square grid. However, because the pixels are 20% closer vertically than horizontally, the circuit shown in figure 8 is used to discard one out of

every 5 lines.

The signal "hblank*" is generated by the video digitizing system.

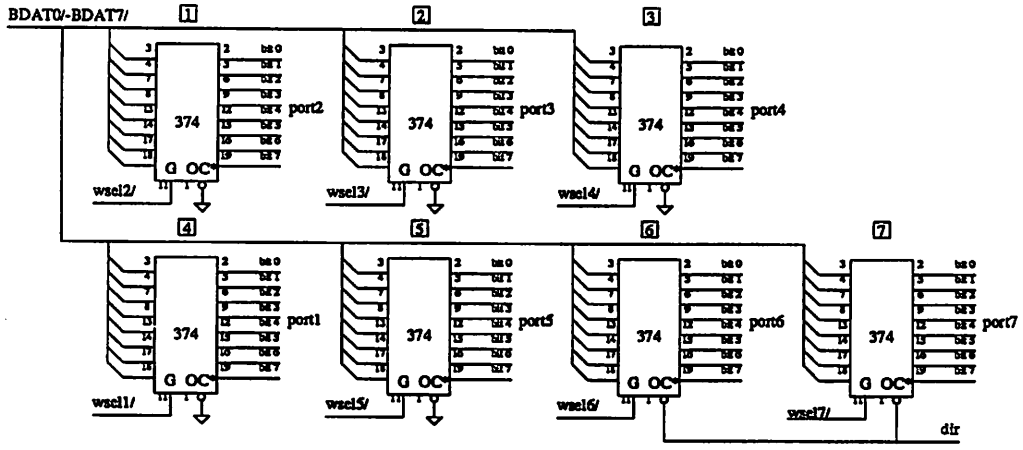



**Figure 5. Latches for Low Rate Control Signals**

The tracer requires a signal that goes high during the frame that is to be loaded into the internal buffer. The circuit in figure 9 generates a signal that is high during the first frame (actually the first even frame after the first odd frame) after the acquire signal is asserted by the host. The timing for this circuit is shown in figure 10. The circuit in figure 9 also synchronizes the "load" signal from the host to clocks on the board.

The video digitizing system generates "vblank*" and "field".

The feature extractor must be synchronized to the internal slow clocks of the tracer. Slow phi1 is brought off the tracer chip but limitations did not allow "slow phi2" to also be brought off. The circuit in figure 11 regenerates the slow clocks from "slow phi1". The timing for the circuit is shown in figure 12. Although the clocks are not identical for the two circuits, there is no overlap between the two phi1 clocks or the two phi2 clocks. The rest of the circuit in figure 11 generates the reset signal for the feature extractor. Because the slow clocks are turned off during the tracer acquire mode, the feature extractor must be reset after every frame has been acquired. In addition, the feature extractor must hold the X and Y start values on the bus for loading by the tracer until these values are loaded into the internal tracer registers. This is why the feature extractor reset is the acquire signal from the tracer delayed by four cycles. The

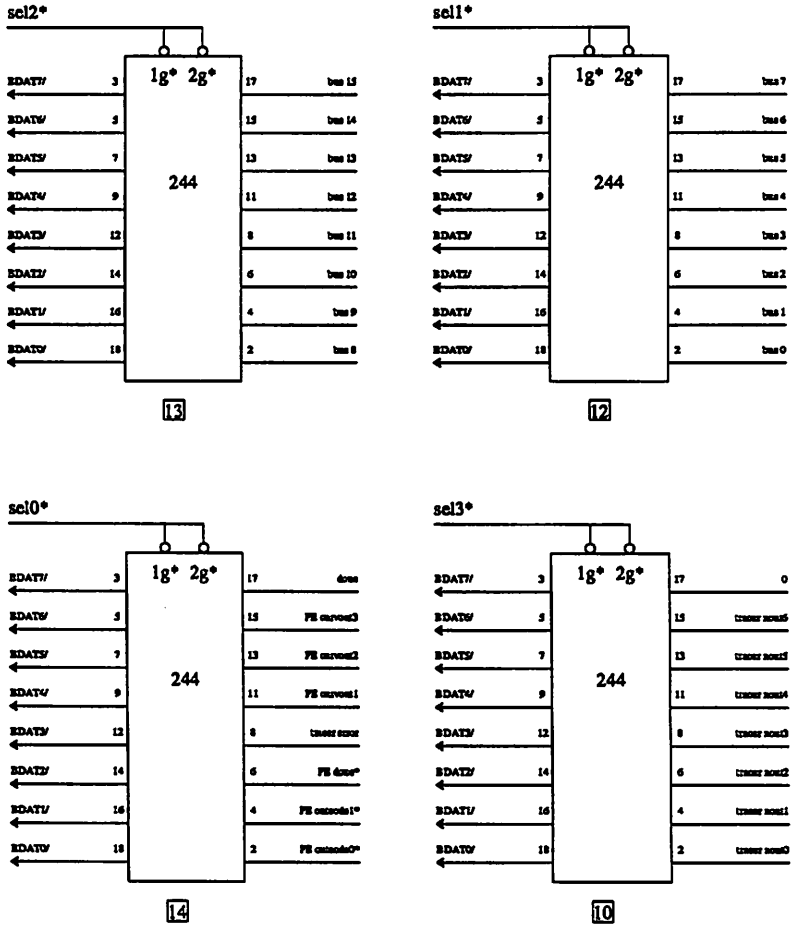



**Figure 6. Buffers for Reading Low Rate Results**

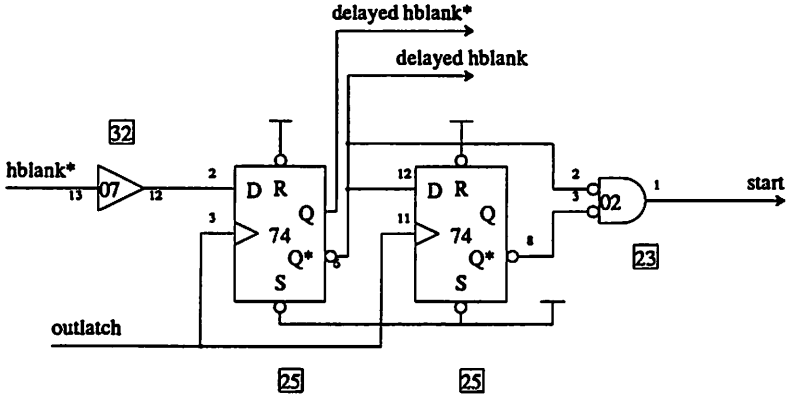



**Figure 7. Start Circuit**

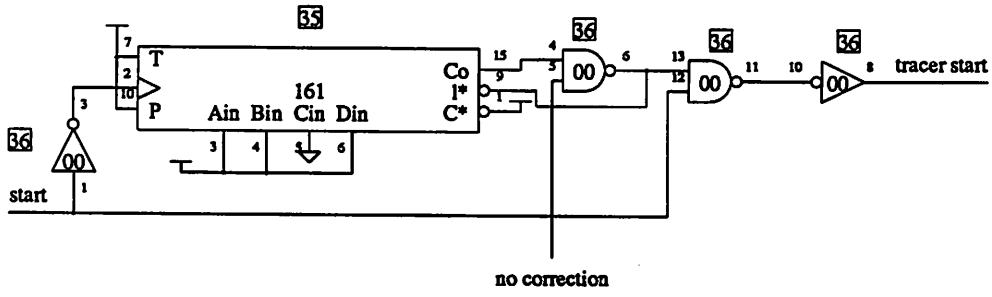



**Figure 8. Contour Tracer "start" Circuit**

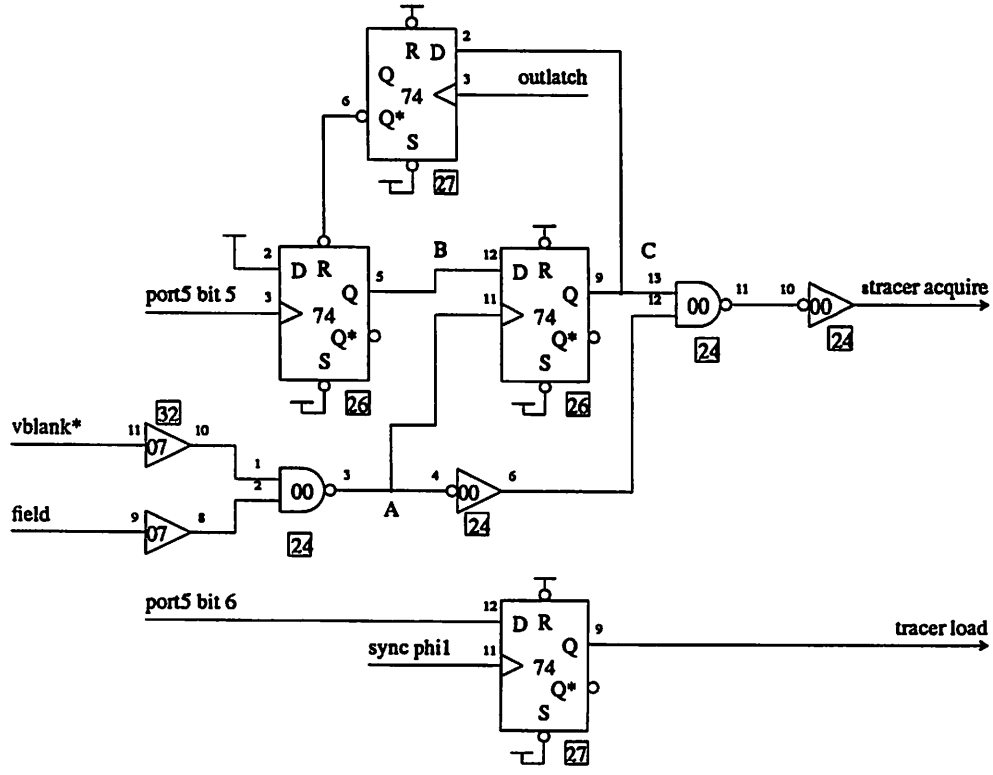



**Figure 9. Tracer "acquire" and "load" circuits**

data from the bus is loaded during the first "slow phi1"

The circuits in figure 13 generate signals for the feature extractor from tracer signals (top two) and handle the buffering of curvature data (bottom). Figure 14 shows the circuit that indicates when a trace is finished. The done signal is reset during acquire and is set when any of the feature extractor status lines is set.

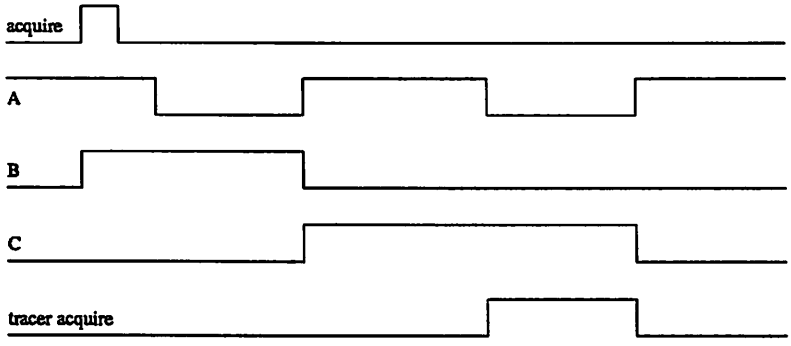



**Figure 10. Tracer "acquire" timing**

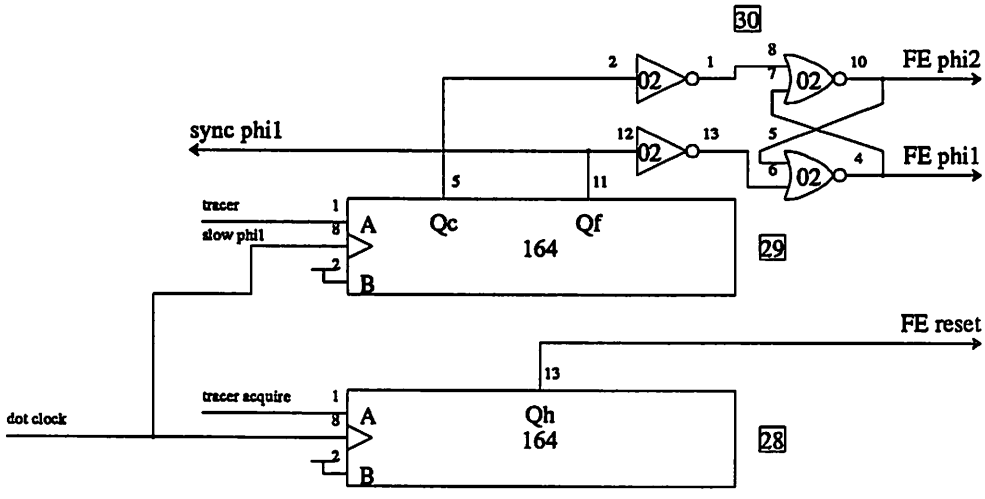



**Figure 11. Feature Extractor Clock Generator and reset Circuits**

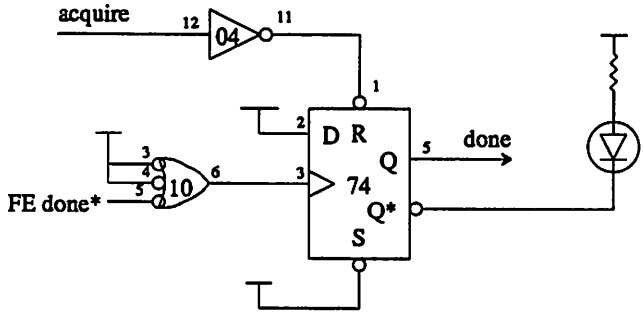



**Figure 14. "Done" Circuit**

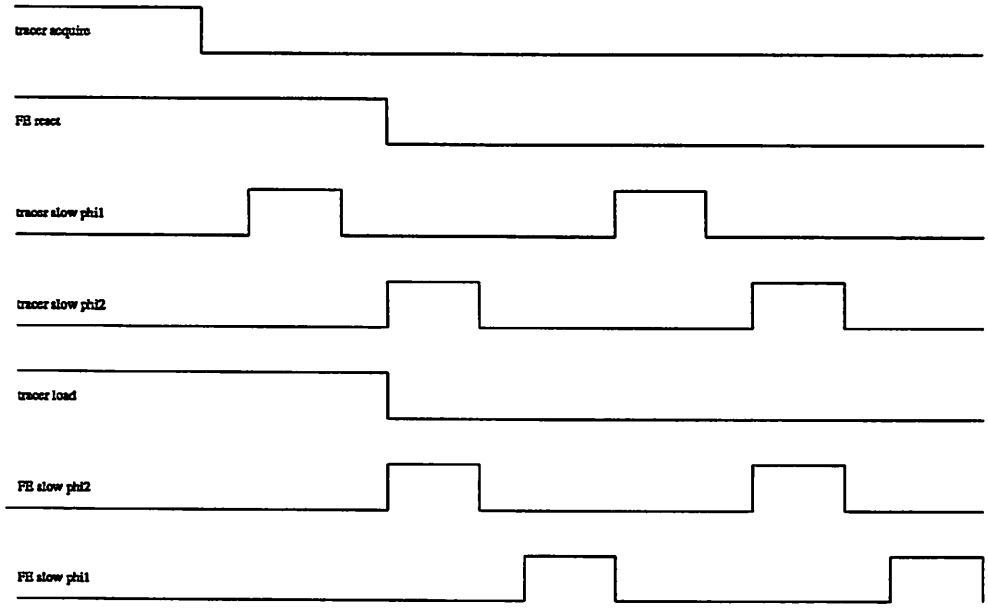



**Figure 12. Feature Extractor Clock Generator and "reset" Circuit Timing**

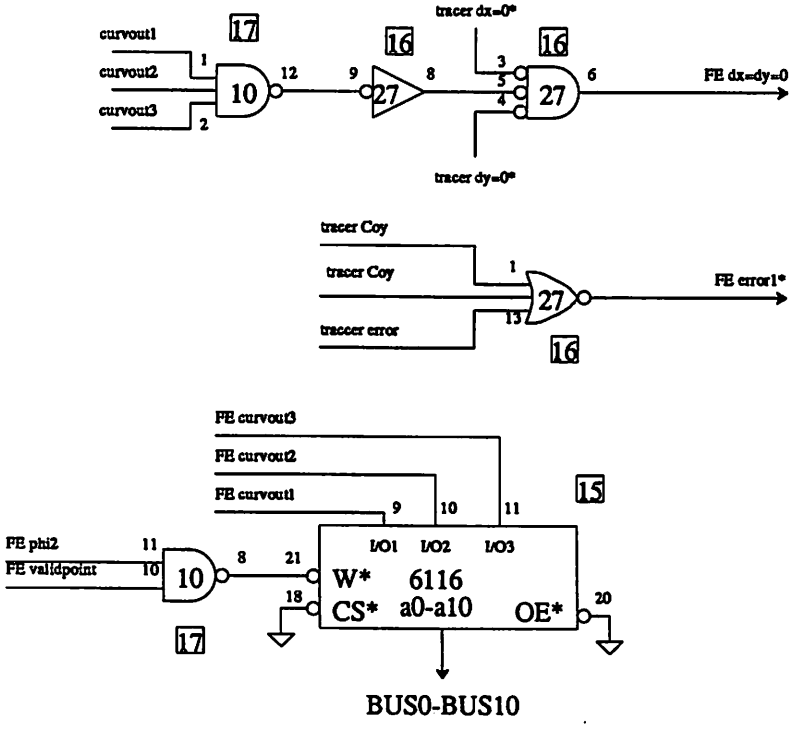



**Figure 13. Feature Extractor Control Circuits**

| Convolver Post Processor | | | | Logical Convolver | | | |
|---|---|---|---|---|---|---|---|
| function | pin | pin | function | function | pin | pin | function |
| sub | 1 | 40 | dout4 | sub | 1 | 40 | din4 |
| dout3 | 2 | 39 | dout5 | din3 | 2 | 39 | din5 |
| dout2 | 3 | 38 | dout6 | din2 | 3 | 38 | din6 |
| dout1 | 4 | 37 | dout7 | din1 | 4 | 37 | din7 |
| dout0 | 5 | 36 | Vdd | din0 | 5 | 36 | bloat0 |
| t7 | 6 | 35 | negclip | bypass | 6 | 35 | bloat1 |
| t6 | 7 | 34 | a7 | dout7 | 7 | 34 | GND |
| t5 | 8 | 33 | b7 | dout6 | 8 | 33 | bloat2 |
| t4 | 9 | 32 | a6 | dout5 | 9 | 32 | bloat3 |
| t3 | 10 | 31 | b6 | dout4 | 10 | 31 | Vdd |
| binarycon | 11 | 30 | a5 | dout3 | 11 | 30 | GND |
| g2 | 12 | 29 | b5 | dout2 | 12 | 29 | phi2 |
| g1 | 13 | 28 | a4 | dout1 | 13 | 28 | phi1 |
| zero a | 14 | 27 | b4 | dout0 | 14 | 27 | |
| absval | 15 | 26 | a3 | start | 15 | 26 | |
| phi1 | 16 | 25 | b3 | | 16 | 25 | |
| phi2 | 17 | 24 | a2 | | 17 | 24 | |
| GND | 18 | 23 | b2 | | 18 | 23 | |
| b0 | 19 | 22 | a1 | | 19 | 22 | |
| a0 | 20 | 21 | b1 | | 20 | 21 | |

**Tables 3,4 Pinouts for the Post Processors and Logical Convolver**

| LUT Rom | | | | Programmable 3x3 Lin Convolver | | | |
|---|---|---|---|---|---|---|---|
| function | pin | pin | function | function | pin | pin | function |
| sub | 1 | 28 | GND | sub | 1 | 40 | dout4 |
| Vdd | 2 | 27 | | dout5 | 2 | 39 | dout3 |
| dout0 | 3 | 26 | | dout6 | 3 | 38 | dout2 |
| dout1 | 4 | 25 | | dout7 | 4 | 37 | dout1 |
| dout2 | 5 | 24 | | start | 5 | 36 | dout0 |
| dout3 | 6 | 23 | Vdd | Vdd | 6 | 35 | GND |
| dout4 | 7 | 22 | GND | GND | 7 | 34 | phi2 |
| dout5 | 8 | 21 | phi2 | | 8 | 33 | phi1 |
| dout6 | 9 | 20 | phi1 | GND | 9 | 32 | GND |
| dout7 | 10 | 19 | din7 | a3 | 10 | 31 | con7 |
| enable | 18 | 30 | din6 | a2 | 11 | 30 | con6 |
| din1 | 12 | 17 | din5 | a1 | 12 | 29 | con5 |
| din0 | 13 | 16 | din4 | a0 | 13 | 28 | con4 |
| din2 | 14 | 15 | din3 | W* | 14 | 27 | con3 |
| | | | | Vdd | 15 | 26 | con2 |
| | | | | din7 | 16 | 25 | con1 |
| | | | | din6 | 17 | 24 | con0 |
| | | | | din5 | 18 | 23 | din0 |
| | | | | din4 | 19 | 22 | din1 |
| | | | | din3 | 20 | 21 | din2 |

**Tables 5,6 Pinouts for the LUT ROM and Programmable Linear Convolver**

| 3x3 Linear Convolver | | | | 3x3 Sorter | | | |
|---|---|---|---|---|---|---|---|
| function | pin | pin | function | function | pin | pin | function |
| sub | 1 | 64 | start | sub | 1 | 64 | din0 |
| Vdd | 2 | 63 | dout7 | start | 2 | 63 | din1 |
| | 3 | 62 | dout6 | Vdd | 3 | 62 | din2 |
| | 4 | 61 | dout5 | GND | 4 | 61 | din3 |
| GND | 5 | 60 | dout4 | | 5 | 60 | din4 |
| | 6 | 59 | dout3 | | 6 | 59 | din5 |
| | 7 | 58 | dout2 | | 7 | 58 | din6 |
| | 8 | 57 | dout1 | | 8 | 57 | din7 |
| | 9 | 56 | dout0 | | 9 | 56 | GND |
| | 10 | 55 | | | 10 | 55 | phi2 |
| | 11 | 54 | | | 11 | 54 | phi1 |
| | 12 | 53 | GND | | 12 | 53 | GND |
| | 13 | 52 | phi2 | | 13 | 52 | |
| | 14 | 51 | phi1 | | 14 | 51 | |
| | 15 | 50 | | | 15 | 50 | |
| | 16 | 49 | | | 16 | 49 | |
| | 17 | 48 | | | 17 | 48 | |
| | 18 | 47 | | | 18 | 47 | |
| | 19 | 46 | GND | | 19 | 46 | invert* |
| | 20 | 45 | sel0 | | 20 | 45 | zeromin |
| | 21 | 44 | sel1 | | 21 | 44 | zeromax* |
| | 22 | 43 | | | 22 | 43 | c3 |
| | 23 | 42 | | | 23 | 42 | c2 |
| | 24 | 41 | sel2 | | 24 | 41 | selmax1* |
| | 25 | 40 | sel3 | | 25 | 40 | c1 |
| | 26 | 39 | din0 | | 26 | 39 | c0 |
| | 27 | 38 | din1 | | 27 | 38 | dout0 |
| | 28 | 37 | din2 | | 28 | 37 | dout1 |
| | 29 | 36 | din2 | | 29 | 36 | dout2 |
| | 30 | 35 | din3 | | 30 | 35 | dout3 |
| | 31 | 34 | din4 | dout7 | 31 | 34 | dout4 |
| din7 | 32 | 33 | din6 | dout6 | 32 | 33 | dout5 |

**Tables 7,8 Pinouts for the Linear Convolver and Sorter**

| Tracer | | | | Feature Extractor | | | |
|---|---|---|---|---|---|---|---|
| function | pin | pin | function | function | pin | pin | function |
| sub | 1 | 64 | yin1 | sub | 1 | 64 | outcode0* |
| yin2 | 2 | 63 | phi2 | outcode1* | 2 | 63 | load |
| yin3 | 3 | 62 | Coxout | done* | 3 | 62 | error1* |
| yin4 | 4 | 61 | dx=zero* | curvout1 | 4 | 61 | dx=dy=0 |
| yin5 | 5 | 60 | dy=zero* | curvout2 | 5 | 60 | Coxin |
| yin6 | 6 | 59 | Coyout | curvout3 | 6 | 59 | reset |
| xin1 | 7 | 58 | start | statein0 | 7 | 58 | curvin1 |
| xin2 | 8 | 57 | slow phi | statein1 | 8 | 57 | curvin2 |
| xin3 | 9 | 56 | yout6 | statein2 | 9 | 56 | curvin3 |
| GND | 10 | 55 | yout5 | statein3 | 10 | 55 | a0 |
| Vdd | 11 | 54 | yout4 | statein4 | 11 | 54 | a1 |
| xin4 | 12 | 53 | yout3 | statein5 | 12 | 53 | a2 |
| xin5 | 13 | 52 | yout2 | xin6 | 13 | 52 | override |
| xin6 | 14 | 51 | yout1 | xin5 | 14 | 51 | Vdd |
| ramout | 15 | 50 | yout0 | xin4 | 15 | 50 | yin6 |
| xout6 | 16 | 49 | selauxram | xin3 | 16 | 49 | yin5 |
| xout5 | 17 | 48 | auxram | xin2 | 17 | 48 | yin4 |
| xout4 | 18 | 47 | statein0 | xin1 | 18 | 47 | yin3 |
| xout3 | 19 | 46 | statein1 | xin0 | 19 | 46 | yin2 |
| xout2 | 20 | 45 | statein2 | bus15 | 20 | 45 | yin1 |
| xout1 | 21 | 44 | statein3 | bus14 | 21 | 44 | yin0 |
| xout0 | 22 | 43 | statein4 | bus13 | 22 | 43 | dir |
| SR out | 23 | 42 | statein5 | | 23 | 42 | bus7 |
| videoin | 24 | 41 | phi1 | | 24 | 41 | bus6 |
| curvout3 | 25 | 40 | load | bus12 | 25 | 40 | bus5 |
| curvout2 | 26 | 39 | acquire | bus11 | 26 | 39 | bus4 |
| curvout1 | 27 | 38 | hsync | bus10 | 27 | 38 | bus3 |
| error | 28 | 37 | phi1w | bus9 | 28 | 37 | bus2 |
| stateout5 | 29 | 36 | phi2w | bus8 | 29 | 36 | bus1 |
| stateout4 | 30 | 35 | GND | validpoint | 30 | 35 | bus0 |
| stateout3 | 31 | 34 | stateout0 | latch | 31 | 34 | GND |
| stateout2 | 32 | 33 | stateout1 | phi2 | 32 | 33 | phi1 |

**Tables 9,10 Pinouts for the Contour Tracer and Feature Extractor**

# APPENDIX B. A COMPARISON OF STORAGE CELLS FOR DIGITAL SIGNAL PROCESSING

## B.1 INTRODUCTION

Every digital signal processing algorithm requires some form of storage for state variables. Every delay in the algorithm corresponds to one word of storage. In image processing, where line delays are used, whole lines (512 or 256 words) must be stored. The type of storage element which yields the optimum tradeoff between power consumption, circuit size and design complexity is very application-dependent.

## B.2 GENERAL TRADEOFFS

Storage can be distributed, such as in a fully parallel implementation, or centralized, as in a micro-coded implementation. If the storage is distributed, every state variable can be accessed each cycle. This is in contrast to the micro-coded case where only a single, or perhaps a few, state variables are needed in each cycle so that the variables can all be stored in a single RAM. The distributed storage elements may be dynamic or static and may have inputs and outputs which are synchronous or asynchronous. Likewise, for the centralized storage elments. Centralized storage has, however, more degrees of freedom. Size of the cell can be traded off for ease of using the cell. As the number of transistors in the cell is decreased, the complexity of the peripheral circuits and the circuit timing increases.

## B.3 DISTRIBUTED STORAGE

Typically, dynamic registers can be used to provide internal distributed storage since clocks are usually available in the system and the inputs and outputs are synchronous to the system clocks. An

example of this is shown in figure 1a, the 6 transistor (T) register. This register will load every clock cycle. If it is desired to hold data in the register, the first clock can be gated with a load signal (figure 1b). Problems with this circuit include: possible clock skew between the gated ph1 and the ph2 clocks (ph1 and ph2 have no overlap). Also, data will remain in the register only as long as leakage currents permit. To remedy these problems, a feedback path can be provided (figure 1c). This circuit will load only when the load signal is high and will refresh otherwise, so that data will remain as long as the clocks run. Also, the original clocks (ph1, ph2) are used to load the 2 halves of the register, so clock skew is not a problem.

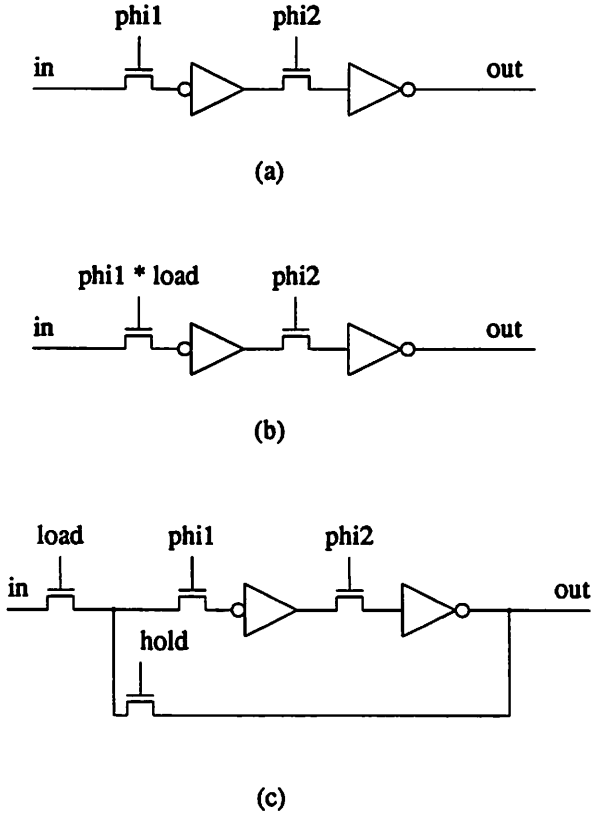



**Figure 1. 6T register (a), load-hold register (b), 7T load-hold register (c)**

## B.4 CENTRALIZED STORAGE

### B.4.1 7T Cell

The choice of cell to use for centralized storage is much more difficult to make. The differences in cell size and peripheral circuit design are great. At one end of the spectrum (largest, but simplest to use)

is the fully static asynchronous write and read 7T cell (figure 2a). This cell has separate read and write bit lines and separate read and write selects, so that read and write operations can occur during the same cycle. The cross coupled inverters refresh each other. The feedback transistor can be a weak depletion device which can be overridden by the write device or an enhancement device that is driven by write select*.

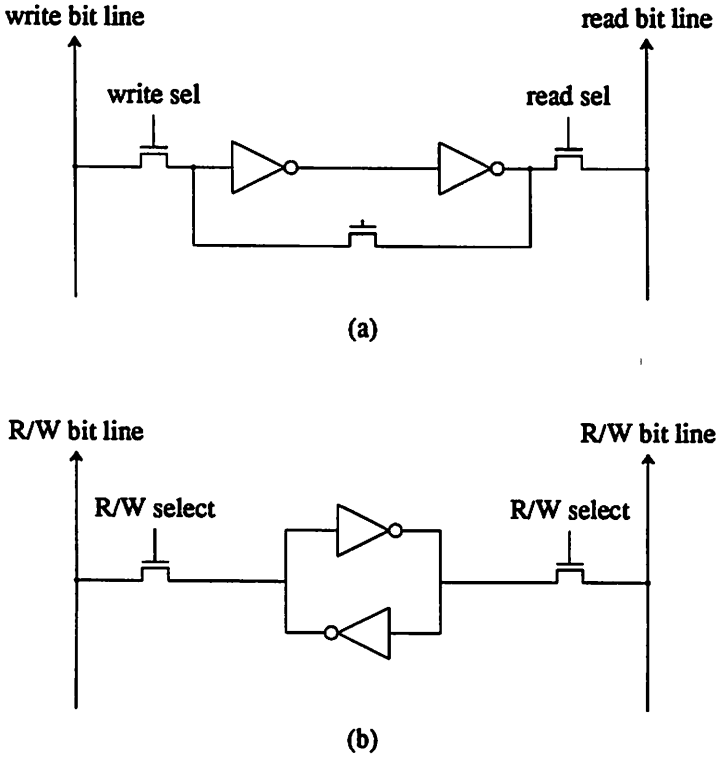



**Figure 2. 7T cell (a), 6T cell (b)**

This cell is useful in small arrays which require static operation, in which refresh cannot be done conveniently, and asynchronous read and write operations. An interface between the internal circuitry and an off-chip processor is one possible application for this cell.

### B.4.2 6T Cell

Another fully static cell is the 6T cell (figure 2b). Again cross coupled inverters store the state of the cell without refresh. However, to get rid of the feedback transistor, both select devices and both bit lines are used to write the cell. The same devices are used to read the contents of the cell. No longer are the read and write operation independent.

### B.4.3 4T Cell

The 4T cell (figure 3a) can be obtained by removing the loads from the 6T cell. This not only makes the circuit consume no static power, but greatly reduces the size of the cell because the depletion devices must have a small W/L ratio (<<1) to minimize power. Without the loads, the circuit is dynamic and must be refreshed. The cross-coupled nature of the circuit automatically refreshes the cell each time the cell is read. This is simpler than having to read and then write the cell to refresh it. To prevent changing the cell contents during a read operation, the select devices must have a small W/L ratio. During a read operation, the bit lines are pre-charged high. If the select devices are large, both nodes in the cell will start to pull up causing a loss of stored information. To ensure this does not happen, the select devices must a small W/L compared to the storage devices. This, however, degrades the cell read time by reducing the cell current. Another constraint imposed by removing the loads is that the read timing must be controlled to ensure that 2 cells are never enabled at the same time. If two cells are enabled one could change the data stored in the other.

### B.4.4 3t Cell

If one of the storage devices is removed from the 4T cell, then one has the 3T cell (figure 3b). The circuit no longer has regeneration and can not be refreshed by simply reading the cell. However, the read and write bit-lines and select lines are now separate so that the read and write operations are independent. Further, the select devices no longer need to be ratioed to prevent changing the cell contents during read operations, because the read select is isolated from the stored charged. More read and write ports can be easily added to this cell. Each additional port requires the addition of a bit line and another select transistor. For the cross-coupled cells, two select devices and bits lines would have to be added.

The 3T cell with separate bit lines has other advantages. Because the write and read data appear on different bit lines, tri-state buffers are not required on the write bit line. Further, it is possible to design a RAM that does not require that the two clocks have any separation between them. In lower frequency applications, clock separation can be obtained by using a higher frequency clock and dividing it down. The clock separation is then defined by some number of the higher frequency clock cycles. This scheme is difficult to utilize in image-processing where the clock rate is high, so that clock separation must be

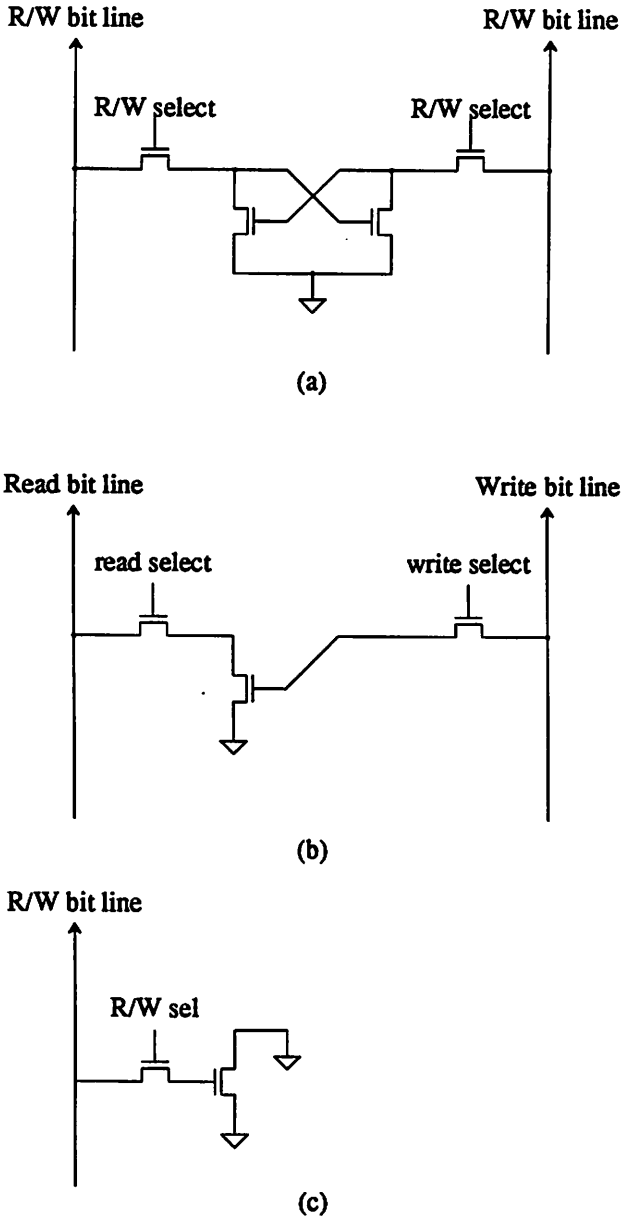



**Figure 3. 4T cell (a), 3T cell (b), 1T cell (c)**

generated by difficult to control circuit delays.

The single bit-line 3T cell requires clock separation while the two bit-line 3T cell does not. With a single bit line, the bit line is usually pre-charged during one clock phase to speed up the read operation. The gated write signal (write enable in the RAM cell) must not overlap this precharge period or the wrong data may be written. To guarantee that the gated write signal does not overlap the pre-charge signal, clock separation must be provided. This is because there is delay from the gate signal (usually a

clock) to the write signal in the RAM. With separate bit-lines the write bit-line is not pre-charged. If the circuit is set up as is shown in figure 5, no clock separation is needed. Essentially, the circuit operates as that shown in figure 6a. The ph1 and ph2 clocks have no overlap, but the gated write signal (ph2 AND W) can overlap either ph1 or ph2 due to delays in the circuit. It can be seen that this circuit will always operate as a single master slave register with no race conditions. However, in this configuration there is only the time of one clock phase to assert the write bit-line and the write select-line. If the circuit were as shown in figure 6b, the bit line would have more time to settle but race conditions would exist, since there is a time when all three register clocks are on. To prevent writing into the wrong cell, the write lines must never go high except when it is desired to write that row. Because the write select is ANDed with ph2, and ph2 always goes low before ph1 goes high, the write line will only go high at the proper time. A timing diagram for this RAM configuration is shown in figure 4.

Separate select-lines makes it possible to read data with very few timing constraints. In the 4T cell, reading and writing occur through the same ports and hence the same care must be taken when reading as with writing to ensure that the cell data is not corrupted. This usually means that the read time must be restricted to less than one clock cycle to ensure that 2 cells are never enabled for reading at the same time. In the 3T cell, the entire cycle can be taken for reading a cell. If two cells are selected for reading at a given time, the contents of the cells are uneffected.

### B.4.5 1T Cell

A simplification that can be made to the 3T cell is to remove the read select device and the read bit-line. This circuit is called a 1T cell (figure 3c), but really has two devices, the select transistor and the storage capacitor. This cell is drastically different from the previous ones. Data is read by directly sensing the charge on the storage capacitor. In all other designs, the output was sensed through some kind of gain stage (a MOSFET). This makes it possible to achieve full logic levels at the output of the cell without any kind of sense amplifier. The 1T cell is read by connecting the storage node to the bit line. Not only does this destroy the stored signal in the cell, but also generates a voltage change which is proportional to ratio of the storage node capacitance to the bit line capacitance (usually $<< 1$). This means that a sense amplifier must be used to restore signal levels. The sense amplifiers add a great deal of area

to the circuit and require complex timing for proper operation.

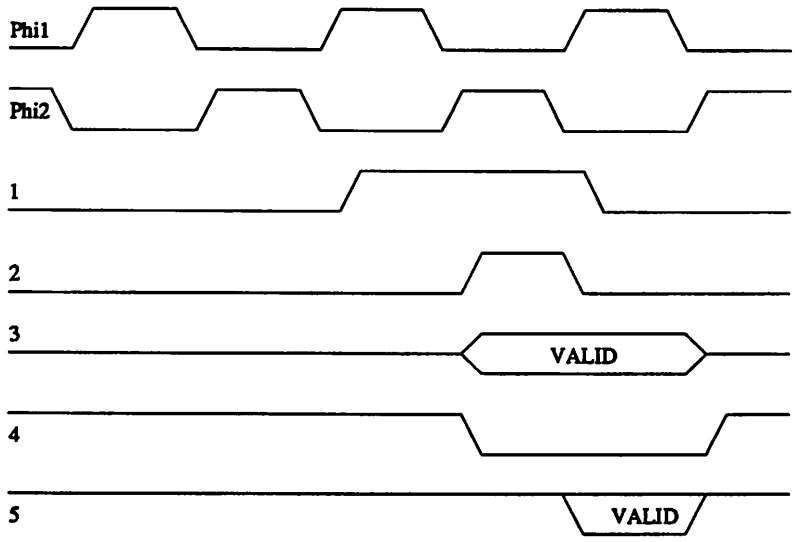



**Figure 4. 3T cell timing**

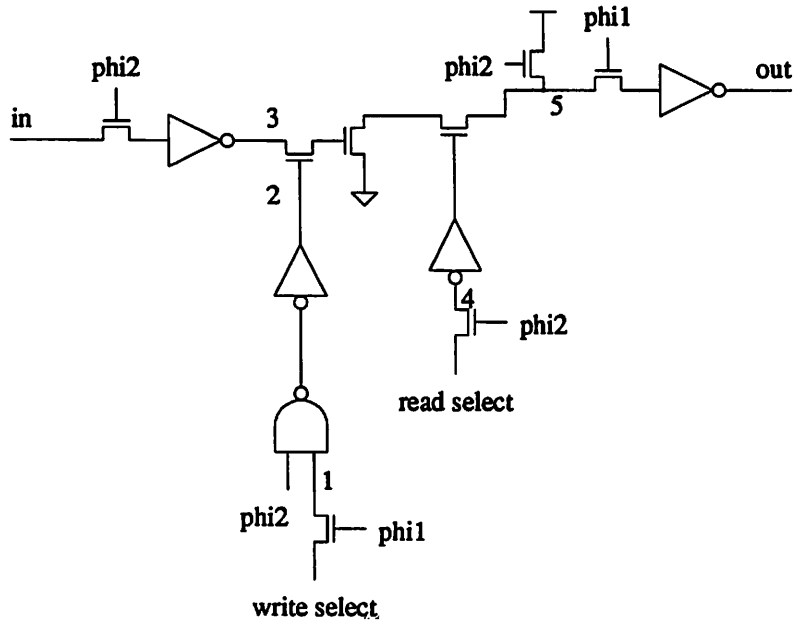



**Figure 5. 3T cell circuit**

## B.5 MAKING CHOICES

Usually, for small storage requirements, the added power and size of the larger cells is not as significant as the added circuit complexity associated with the smaller cells.

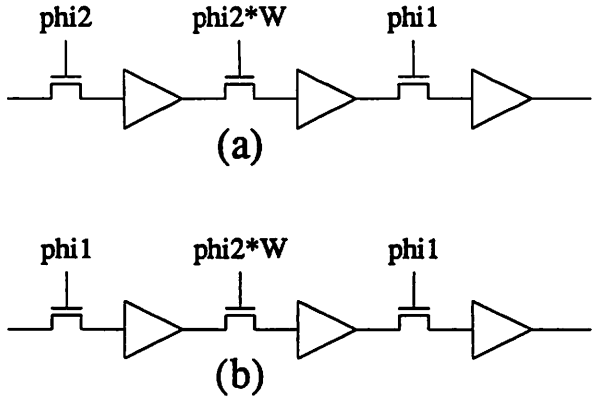



**Figure 6. Equivalent Circuits**

Often a small (e.g. 3T) cell can be used with little design penalty. For example, if the RAM locations are are written at well defined periods that are less that the refresh times, a dynamic memory will work just like a static memory. Sense amplifiers can often be avoided, even when working with arrays of 2K bits at 10 MHz.

If static operation is required with a large RAM array, it is likely that a dynamic RAM with refresh will be more cost effective than a static RAM. This is because, circuitry must be added to each cell to make the cell static, but the refresh circuitry is only added to each row and/or column. Further, the power consumption will increase as R+C (number of rows plus the number of columns) for the refresh circuitry and by RC for the static cells. To perform the refreshing without slowing the processing down, one can often utilize specific points in the algorithm where the RAM would be inactive.

Sense amplifiers are rarely needed with any of the RAMs except the 1T RAM. This is because the select line pull-up times are the largest part of the total propagation delay. To minimize the size of the cell, typically, the select lines are poly and the bit lines are metal. The poly select lines have a large capacitance in the form a gate capacitance while the bit line capacitance is composed of the metal to bulk and active area capacitances. The poly has a much higher sheet resistance than metal so that the RC delays in the select lines is often significant. The bit lines can be pre-charged quickly and even with very large bit lines (5mm), the RAM cell can pull the bit line down quite fast. If sense amplifiers could make the bit line delay approach zero, the total RAM propagation delay would not change by much. Of course, this situation would change if the select lines were metal and the bit lines were active areas.

## B.6 DESIGN CONSIDERATIONS

### B.6.1 Pre-charged Bit-Lines

Typically, to achieve high speed in a clocked system, the bit lines are pre-charged high. Care must be taken to ensure that the off-current is small for all cells which are not enabled but connected to the same bit line. This is particularly true for large arrays. If many cells are connected to the same bit line, each with a small leakage current, the total current may be large enough to pull the bit line down when not desired. There are two techniques to avoid this problem. First, a static pull-up can be connected to each bit line to provide a source of current so that the leakage currents will not pull charge of the bit line capacitance. Ratioed logic must be used when static pull-ups are employed. To keep the size of the memory cell small, it is desirable to keep the static pull-up current small. Using weak pull-ups also reduces the power dissipation. The second technique is to ensure that select lines pull down as far as possible to reduce the leakage current in each cell. Care should be taken to ensure that the ground potential in the select driver does not rise above that of the array. If the driver ground is positive with respect to the array ground, this will show up as positive bias on the select device, increasing its leakage current. Further, larger W/L ratios should be used in the pull-down device in the driver than is used in standard logic to provide a lower output low voltage. If a push-pull driver is used with enhancement pull-ups, the output low voltage is quite low.

### B.6.2 Maintaining Long Storage Times in Dynamic Memories

Similar problems exist in maintaining stored charge on the storage node of a dynamic memory cell. To obtain long refresh times, leakage currents should be kept small. This means keeping diffused regions which connect to the storage node small, since the reverse bias leakage current is one component of the total leakage current. Charge may also leak through the select device if it is not turned off completely. To keep this component small, the write select drivers should pull down near zero volts and a good ground potential in the drivers should be maintained. Signals which are capacitively coupled to the storage node can also change the voltage stored on the storage capacitor. Care should be taken not to cross the storage node with signals which may alter the stored signal significantly.

Lab tests can help indicate which of these problems exist. Increasing the substrate bias (more negative bias on a p-type substrate) will increase the threshold of the enhancement select devices and decrease the leakage current. At the same time, the width of the depletion regions will increase as will the leakage current due to space-charge generation. If the refresh time increases with substate bias, the drivers are likely the problem. If it decreases, p-n junction leakage could be the problem. Another test is to increase the $V_{DD}$ supply. This will raise the voltage at the gates of pull down transistors in the select drivers. Because the current through this device varies nearly linearly with gate bias in the triode region (for constant $V_{DS}$) and the depletion device current is only a weak function of the supply voltage (typically, the device will be in saturation when the select line is low and changes in $V_{DD}$ will affect the current only through channel length modulation effects), the net affect seen by raising $V_{DD}$ is a decrease in the output low level of the select driver. This results in a decrease in the leakage current through the select device.

### B.6.3 Example of RAM problems

A 4K RAM using a 3T cell with separate bit lines was constructed for use in the delay line. The bit lines are pre-charged since there is a large capacitance due to the metal-oxide-bulk capacitance and the diffused capacitance. Simulations showed that a static pull-up ratioed (with the RAM cell) to provide a good output low level could not appreciably change the bit line voltage in the cycle times that the RAM was to operate. It was thought that the bit line capacitance was large enough that the static pull-ups were not needed. The circuit did not work. The errors which occurred were very data-dependent, but it could be seen that the bit line was pulled down when it should not be. Some tests showed:

1. Increasing the pre-charge time helped the problem. This showed that the problem was likely at the read bit line as suspected, since the bit line has 64 devices connected to it.

2. Running the circuit slower made the problem worse. This tended to indicate that leakage currents of some kind were the problem, since a fixed current would be able to make a greater change on the voltage on a capacitor in a longer time.

3. An array with half sized bit lines also made the problem worse. This seemed to indicate that the

problem is not reverse biased p-n junction leakage since the number of leakage current sources halved as the capacitance halved. Therefore, the effects of junction leakage currents should be constant with a change in length of the RAM bit lines.

4. Increasing $V_{BB}$ (more negative) did not help. Negative substrate bias tends to slow the circuit down by decreasing the currents in the devices while only decreasing the junction capacitance and not the oxide capacitance. Because of this it was not obvious which effect was being seen, a change in the leakage currents or a change in the speed performance of the circuit.

5. Shining light on the substrate fixed the problem! This was quite a surprise, since it was suspected that leakage currents of some type were to blame. Shining light on the die generates free carriers in the substrate which look very similar to leakage currents of the reversed biased junctions. The electrons are collected by the n+ regions. At the bit lines a net current flows out of the bit line. This would tend to pull the bit line down not up.

6. Putting positive bias (~.4V) on the substrate fixed the problem. Positive substrate bias has several effects. First, the thresholds of the devices are lowered so that leakage through the select devices should increase. The depletion region widths decreases, decreasing the junction leakage current. Also, it injects electrons into the substrate and holes into the n+ regions at ground potential. The electrons in the substrate will behave as those created by shining light on the die. No holes will be injected into the n+ regions attached to the bits lines as these regions are at a potential far above ground when pre-charged (no problems were observed on bit lines that should be pulled down). It appears that shining light on the die and putting positive substrate bias on the circuit have the creation of free electrons in the substrate in common.

No simple theory could explain all these facts. All other memories (except one) included the static pull-ups on the bit lines and worked. The circuit was changed to include the pull-ups and the circuit worked fine with still no explanation as to what the problem was.

# APPENDIX C NMOS CIRCUIT LAYOUT GUIDELINES

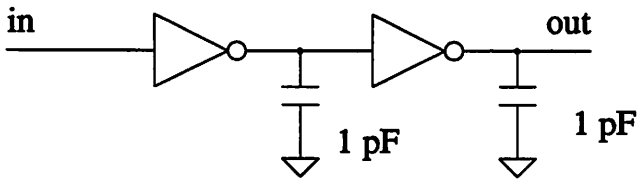
## C.1 INTRODUCTION

"VLSI" circuit design has been talked about by many [1]. Usually attention is paid to implementing the desired logic functions with the fewest and most regularly arranged transistors and circuit blocks. The electrical characteristics of the transistors and the interconnect are usually ignored for the sake of simplicity. However, in reality the transistors do not act as perfect switches and the interconnect layers are not perfect conductors. Certain guidelines should be followed when designing circuits which take these attributes into account to prevent circuits from being designed that are correct logically but not electrically. Further, to reduce the sensitivity of the circuit performance to processing variations, the user should design critical parts of the circuit carefully.

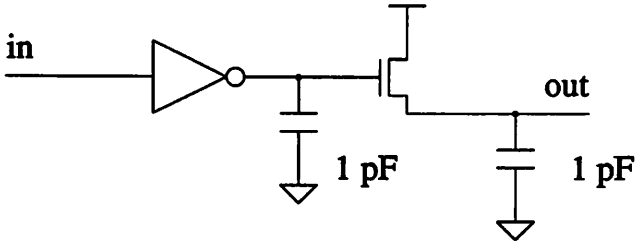
## C.2 TRANSISTORS

The enhancement devices are very different from ideal switches under certain circumstances. Likewise, the depletion devices characteristics can vary greatly. The enhancement device operated in the common source configuration operates very nearly as a switch. When the gate is high, the drain-source impedance is low. When the gate is low the drain-source impedance is very large. A common source transistor with a depletion pull-up produces output voltages that can swing close to $V_{DD}$ and ground. When the enhancement transistor is used as a pull-up device the source is raised above ground and the device operation changes. This situation routinely occurs when enhancement devices are used as pass gates or dynamic pull-ups. The logic output levels are a strong function of both the device zero bias threshold voltage and the body effect parameter, $\gamma$. Because the device turns off when $V_{GS}-V_t=0$ the source can only rise to $V_G-V_t(V_S)$. When the circuit is operated from a single supply, these nodes can only pull-up to $V_{DD}-V_t$. The actual voltage that is obtained is process-sensitive and is a function of both

$V_{to}$ and $\gamma$.




CIRCUIT 1




CIRCUIT 2

**Figure 1. Circuits Used in Transient Simulation**

The transient behavior of source-follower circuits is quite different from a common-source circuit and must be treated accordingly. The source-follower has the disadvantages that both the input and output are rising in the worst case and NMOS devices pull-up much slower than they can pull-down. Further, the gain of the circuit is less than one so the input to the source follower must rise very close to $V_{DD}$. The net result of these disadvantages is that the source follower must have a higher performance driver than the common source circuit. The circuits shown in figure 1 were used to illustrate this point. All enhancement devices were chosen to be the same size (W/L=8/4) and the depletion devices were chosen to give a good output low level. Spice simulation indicate that the output of circuit 1 reaches 0.5 V before the driver reaches 3.75 V (120 nSec). For circuit 2, however, the output voltage lags behind the driver voltage. The output of circuit 2 reaches 2.5 V after 185 nSec and 3.0 V after 235 nSec. After 300 nSec, the output of circuit 2 only reached 3.3 V.

Depletion devices are used at times to allow poly to cross the active area without creating an

enhancement device. The depletion device is assumed to be low impedance regardless of the gate voltage and hence to act as a simple interconnect crossing. The parasitic depletion device in a pull-down path decreases the pull-down current, but can be compensated for by decreasing the pull-up device current. If the depletion device has to pull-up, the output voltage will only rise to $-V_{tD}$ (the depletion device threshold voltage) with zero gate bias. For depletion devices with thresholds in the -2.5 V range, this type of circuit should be used only with care.

The enhancement device operated as a pass gate has logical characteristics which are quite different from an enhancement device operated in a common-source configuration. A common-source enhancement device is basically a uni-directional device. That is, voltage at the gate of the device strongly affects the drain current but the drain current and voltage have little effect on the gate voltage or current. When operated as a pass-gate, the device is bi-directional because the source and drain are identical. In some cases this is a desired characteristic and can be used to the designers advantage. However, it is possible that the bi-directional nature of the device will cause problems if undesired signals propagate in the direction opposite to desired signals. This can happen in multiplexor circuits made of 2 pass gates with logic to turn one of the two transistors on. If the turn-on and turn-off times of the drivers are asymmetric, the two pass-transistors can be turned on at the same time. It then becomes possible for signals to propagate from one input to the other. This will not necessarily cause errors but the designer should be aware of the possibility. If the inputs were connected only to MOSFET gates, the transmission between inputs would be virtually eliminated.

## C.3 INTERCONNECT

Interconnect (typically poly, active area and ,metal) has several properties that make it a non-ideal conductor. Most importantly, the interconnect layers have both resistance and capacitance. The diffused regions also form p-n junctions with the substrate. These p-n junctions result in two effects. First, if the p-n junction is forward biased when a signal goes below ground, electrons will be injected into the substrate to be collected elsewhere. When the p-n junction is reversed biased it collects electrons from the substrate and those generated in the space charge region.

Since dealing with the capacitance is of primary concern in most circuit design courses and "VLSI" design books, the resistance of the interconnect is probably the most important aspect that is ignored. In general the metal layer provides the lowest resistance ($\tilde{}40\ \frac{m\Omega}{sq}$) and the lowest capacitance. These properties make it the preferred conductor. Low resistance is important when DC current flows and low ohmic voltage drops are desired. Even when there is no DC current, low resistance is desirable to minimize RC delay times on long wires with a significant capacitance. Metal should be used for all (except for very local) $V_{DD}$ and ground wiring to minimize ohmic drops. It is important to run the system clocks in metal to minimize RC delays that can result in clock skew or overlap. There is usually no DC current in the clock lines but the clock signals should not overlap after propagating through the interconnect. The resistance of poly or the active area can cause excessive delay in one of the clocks and cause the clocks to overlap. Of course, if the circuit is perfectly symmetric, the clock signals will be delayed equally and no clock skew will develop.

Poly and the active area should be used for short or low speed signals. If the signal lines are long and high speed is desired, the resistance ($K\Omega s$ of resistance are not uncommon in poly or diffused lines longer than 100 microns) and capacitance of the two layers must be considered. In NMOS, care must also be taken when a signal line has DC current flowing in it to ensure that ohmic voltage drops are not too large. This situation can arise when the pull-up of a logic gate is connected to the pull-downs through a poly line.

The p-n junctions associated with the diffused regions can be a problem in certain circumstances. If signals from off-chip are not buffered (such as clocks) and are connected to diffused regions, the p-n junctions can be forward biased by negative going swings of the signal. The electrons injected into the substrate when the junction is forward biased will be collected at other nodes in the circuit possibly causing errors. Diffused regions connected to dynamic storage nodes collect electrons which tend to neutralize the stored charge. Although it is necessary to have the storage node connected to a diffused region to form the pass-gate, the area of the diffused region should be minimized.

## C.4 GENERAL CHIP LAYOUT

When a complete chip is put together from macrocells, the supplies, clocks and signals should be routed in an order that minimizes undesirable effects. First, the supplies should be routed in metal to guarantee that no cross-overs will be needed. The width of the $V_{DD}$ and ground wires should be chosen to keep the ohmic voltage drops below a couple tenths of volts and to prevent metal migration. The latter usually requires that the conductors carry no more than 1 $\frac{mA}{\mu}$. Next the clocks, starting with the fastest clocks and proceeding to the slowest clocks, should be routed in metal and should jump only the supplies or higher speed clocks in poly. The width of the poly cross-overs should be chosen to minimize RC delays. Further, the cross-overs should be kept very similar so that one clock is not delayed more than the other. The resistance of the cross over should be reduced as the capacitance of the line following the cross over increases. If the clocks are routed only in poly or metal, there is no chance that charge will be injected into the substrate if an off-chip clock driver circuit produces a clock which rings below ground. Finally, all remaining signals can be routed in metal with poly or diffusion cross-overs for crossing the supplies, clocks and other signals.

## C.5 SENSITIVITY

In NMOS circuits the power consumption and speed of the circuit are both strongly dependent upon the depletion device $K_p$ and effective width. In the static parts of the circuit, the power consumption is almost solely determined by the depletion device current and $V_{DD}$. When a node is pulled-down, the depletion device is typically saturated and hence the DC current is only a weak function of the enhancement $K_p$ but a linear function of the depletion $K_p$ and width. This is because changes in the enhancement device $K^p$ will change the output low voltage which has only a small effect on the depletion device current through channel length modulation effects. The speed of the circuit is also a strong function of the depletion device current because the pull-up time is usually much greater than, typically four times, the pull-down time. Therefore, if the enhancement currents are halved, the total delay time (i.e. $\tau_{LH}+\tau_{HL}$) is increased by only 11%. If the depletion currents are halved, the total delay is increased by 66%.

Because the circuit power dissipation and speed are very dependent upon the depletion device currents, it is desirable to make these currents as insensitive as possible to processing variations. The designer can do little about changes in $K_p$ but can reduce the sensitivity to depletion device width. The use of minimum width depletion devices makes the depletion device current more sensitive to processing. This is because the depletion device width is reduced by the encroachment of the field regions and this encroachment has a greater relative affect for small device widths. Therefore, the designer can use greater than minimum width devices in the time critical parts of the circuit to reduce the sensitivity of the circuit speed to processing. Because the time critical parts of the circuit usually require large currents for small rise and fall times, the power consumption and delay times will both tend to stabilize.

## C.6 REFERENCES

[1] Mead, C., Conway, L., *VLSI Systems*, Addison Wesley, Reading, Mass., 1980.

# APPENDIX D CAD INPUT DESCRIPTIONS

## D.1 INTRODUCTION

This section contains the CAD descriptions which were used to generate various parts of the chips. These are the actual descriptions that were used and not the descriptions that would be used with upgraded versions of the programs.

Some parts of the linear convolver, contour tracer, feature extractor, non-linear post processor, LUT ROMs and sorting filter were generated automatically.

## D.2 SORTING FILTER

The data-path descriptions for the two data-paths used in the sorting filter are given in listings 1 and 2. These descriptions are for an old version of the generator that did not deal with control nets on the top and bottom of the data-path.

## D.3 Feature Extractor

The three data-paths and the three PLAs used in the feature extractor were generated automatically with the data-path generator and Modgen, respectively. Listings 3 to 8 show the input descriptions for these circuits.

## D.4 CONTOUR TRACER

The FSM ROM in the contour tracer was generated from a state transition table (listing 9). A special program was written to convert this table to the binary ROM listing. The program generates some of the ROM outputs that are more easily specified by simple logic in the program than in the state table.

The high three bits of the state numbers in general correspond to the current trace direction and the

low three bits correspond to the test direction. Some exceptions are the scanning state (3), refresh states, the error state and the initial states after scanning. Output value "4" corresponds to an error output.

### D.5 Linear Convolver

The micro-code for the linear convolver arithmetic controller was generated from human readable impulse responses (listing 10). A special purpose program was written to perform this task. The program determines the binary control signals from the constant definitions given at the top of the input file. It also checks to see if arithmetic overflow could occur.

### D.6 Non-Linear Low-Pass Filter

The data-path generator input file for the non-linear low pass filter is given in listing 11.

### D.7 LUT ROMs

The LUT ROMs were generated automatically from a desired function (e.g. logarithmic). The problem was broken up hierarchically. A program at the lowest level assembled a ROM directly from a binary bit pattern for the desired number of rows and columns. A second program generates this bit pattern from a list of desired binary output words. This would be the actual bit pattern if column decoding were not employed. Finally, a third program generated the list of output words from a desired function.

This approach makes it possible to enter the system at different levels to suit the designer.

**Listing 1. Maximum-Median Data-Path Description**

```
*
      maxmed -- compute maximum or medean over 3x3 region

      7/16/85
*
LEFT 'in1' 'in2' 'in3'   * inputs *
RIGHT 'max'              * output -- max or med *

ORGANIZATION
      CNT      NODATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      GND      NODATA

SLICE

      * compute vertically *

      'in2' > minus1 > minus1 > zero

      > sort2 ('inb'='in1','outb'='min1','out'='max1')

      'in3' > delay > delay >zero

       > sort2 ('inb'='max1','out'='maxout','outb'='min2')

      'min1' > delay > delay

      > sort2 ('inb'='min2') > mux2to1 ('inb'='maxout') >

      * compute horizontally *

      delay ('out'='max2')> delay ('out'='max3') > delay > zero

      > sort2 ('inb'='max3','outb'='min3','out'='max4')

      'max2' > delay > delay > zero

       > sort2 ('inb'='max4','out'='max5','outb'='min4')

      'min3' > delay > delay

      > sort2 ('inb'='min4') > mux2to1 ('inb'='max5') > buffer > 'max'
```

**Listing 2. Minimum Data-Path Description**

```
*

 min circuit -- performs minimum over 3x3 area (or sub rectangle)
              also has post processing to get max-min

       inputs from 3 different lines

       7/16/85 *

LEFT 'in1' 'in2' 'in3'
RIGHT 'out' 'max'

ORGANIZATION
       CNT      NODATA
       ODD      DATA
       EVEN     DATA
       ODD      DATA
       EVEN     DATA
       ODD      DATA
       EVEN     DATA
       ODD      DATA
       EVEN     DATA
       GND      NODATA

SLICE
       * first compute min over vertical areas *

       * compute min (in1,in2) *
       'in2' > minus1 > zero > minus1 > sort2 ('inb'='in1','outb'='min0')

       'min0' > minus1 > 'min1'              * invert min *

       'in3' > delay > delay > minus1 > zero       * invert in3 *

        > sort2 ('inb'='min1')          * inverted min (in1,in2,in3) *

       * compute min over horizontal region *

       > delay ('out'='mindelay1') > delay('out'='mindelay2')

       > delay > zero > sort2 ('inb'='mindelay2') > 'min3'     * first min*

       'mindelay1' > delay > delay > zero > sort2 ('inb'='min3') * second min*

       * no invert to get min, invert to subtract*

       > minus1 > varminus1 > delay > 'min'

       'max' > delay > zero > adder ('inb'='min') > delay

       > buffer > 'out'
```

**Listing 3. Main Data-Path Description used in the Feature Extractor**

```
*
      Input description for tracer post processor that
      computes perimeter, odd perimeter, x average, y average
      and the area.

      Requires 16 bits

      8/12/85
*

LEFT      'bus' 'x'
RIGHT     'bus' 'y'

ORGANIZATION

      CNT      NODATA
      MSB      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      GND      NODATA

SLICE

      *perimeter, odd perimeter*

      'perimeter' > increment > delay-clear ('out'='perimeter') > TSbuffer
      > 'bus'

      'oddperim' > increment >  delay-clear ('out'='oddperim') > TSbuffer
      > 'bus'

      * 1st moments of periphery *

      'x' > shiftright > delay > zero > posadder ('inb1'='xb')
      > possataccum ('out1'='xb') > zero > TSbuffer > 'bus'

      'y' > shiftright > delay > zero > posadder ('inb1'='yb')
      > possataccum ('out1'='yb') > zero > TSbuffer > 'bus'
```

```
* area *

zero-one > 'temp'
'y' > delay > adder1 ('inb'='temp') > absvalue > zero  >
adder ('inb'='area') > delay-clear ('out'='area') > TSbuffer > 'bus'
```

**Listing 4. "X" Data-Path Description used in the Feature Extractor**

```
*
      Input description for tracer post processor part
      that computes xmin, xmax and holds xstart.
      requires 7 bits.

      8/12/85
*

LEFT 'bus' 'x'
RIGHT 'bus' 'x'

ORGANIZATION

      CNT      NODATA
      MSB      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      ODD      DATA
      EVEN     DATA
      GND      NODATA

SLICE

      *xmax*

      'x' > delay > zero > compare ('inb'='max','outb'='maxtemp') >
      maxselect ('inb'='maxtemp') > invert-clear > delay > 'max' > increment
      > TSbuffer > 'bus'

      *xmin*

      'x' > delay > minus1 > zero > compare ('inb'='min','outb'='mintemp') >
      maxselect ('inb'='mintemp') > invert-clear > delay ('out'='min') > minus1
      > TSbuffer ('out'='bus')

      *xstart*

      > buff > latch ('outg'='bus')
```

**Listing 5. "Y" Data-Path Description used in the Feature Extractor**

```
*
      Input description for tracer post processor part
      that computes ymin, ymax and holds ystart.
      requires 7 bits.

      8/12/85
*

LEFT 'bus' 'y'
RIGHT 'bus' 'y'

ORGANIZATION

      CNT     NODATA
      MSB     DATA
      EVEN    DATA
      ODD     DATA
      EVEN    DATA
      ODD     DATA
      EVEN    DATA
      ODD     DATA
      EVEN    DATA
      GND     NODATA

SLICE

      *ymax*

      'y' > delay > zero > compare ('inb'='max','outb'='maxtemp') >
      maxselect ('inb'='maxtemp') > invert-clear > delay > 'max' > increment
      > TSbuffer > 'bus'

      *ymin*

      'y' > delay > minus1 > zero > compare ('inb'='min','outb'='mintemp') >
      maxselect ('inb'='mintemp') > invert-clear > delay ('out'='min') > minus1
      > TSbuffer ('out'='bus')

      *ystart*

      > buff > latch ('outg'='bus')
```

**Listing 6. Modgen PDL file for FSM used in feature extractor**

```
(module
    /* tracer post processor FSM

      keeps track of state of tracer and generates some
      basic outputs. The tracer outputs are delay for
    writing into the RAM.
    */
    (name fsm )
    (type fsm )
    (in 13 )
    (out 14 )
    (minterm 25 )
    (numclocked 12)
    (output-plane
        (array 00000000100111)
        (array 00000000000011)
        (array 00000000100110)
        (array 00000000001101)
        (array 00000000001001)
        (array 00000000001011)
        (array 00000000000000)
        (array 00000010100100)
        (array 00000010100101)
        (array 00000000000000)
        (array 00000010101100)
        (array 00000010101001)
        (array 00000010111110)
        (array 00000000111100)
        (array 00000100110111)
        (array 00000110110100)
        (array 00000100110100)
        (array 00000000111010)
        (array 01000001000000)
        (array 01000001000000)
        (array 01000001000000)
        (array 01000001000000)
        (array 00100000000000)
        (array 00010000000000)
        (array 00001000000000)

        /* outputs (left to right):
            dummy
            validpointnc*
            out2
            out1
            out0
            resetreg*
            done*
            validpoint*
            selstart*
            outcode1*
            outcode2*
```

```
                FSM state 0
                FSM state 1
                FSM state 2
            */
        )
        (input-plane
            (array 000x0xx00xxxx)
            (array 000x1xxx0xxxx)
            (array 000x0xx10xxxx)
            (array 001xx11x0xxxx)
            (array 001xx0xx0xxxx)
            (array 001xx10x0xxxx)
            (array xxxxxxxxxxxxx)
            (array 0101xxxx0xxxx)
            (array 0100xxxx0xxxx)
            (array xxxxxxxxxxxxx)
            (array 0111xxxx0xxxx)
            (array 0110xxxx0xxxx)
            (array 100xxxxx0xxxx)
            (array 101xxxxx0xxxx)
            (array 11000xxx0xxxx)
            (array 11001xxx0xxxx)
            (array 1101xxxx0xxxx)
            (array xxxxxxxx1xxxx)
            (array 001xxxxx00xxx)
            (array 001xxxxx0xxx0)
            (array 001xxxxx0x0xx)
            (array 001xxxxx0xx0x)
            (array xxxxxxxxxxx0x)
            (array xxxxxxxxxx0xx)
            (array xxxxxxxxx0xxx)

            /* inputs (left to right):
                FSM state 2
                FSM state 1
                FSM state 0
                load
                scanningin*
                error1*
                dx=dy=0
                cox
                reset
                out0
                out1
                out2
                state=36*
            */
        )
    )
```

**Listing 7. Modgen PDL file for control PLA used in feature extractor**

```
(module
    /* tracer post processor PLA.
      computes information from tracer state
     and validpoint.
    */
    (name pla1 )
    (type fsm )
    (in 7 )
    (out 8 )
    (numclocked 6)
    (minterm 9 )
    (output-plane

        (array 10000000)
        (array 01000000)
        (array 00000001)
        (array 00000001)
        (array 00000010)
        (array 00001000)
        (array 00000100)
        (array 00001000)
        (array 00010000)

        /* outputs (left to right):

            scanningin*
            state=36*
            dummy
            validoddpoint*
            0*
            1/2*
            -1/2*
            +1*
        */
    )
    (input-plane

        (array x000011)
        (array x011110)
        (array x01xxxx)
        (array x10xxxx)
        (array x1x1xxx)
        (array xx10xxx)
        (array x0x1xxx)
        (array 1xxxxxx)
        (array 0xx0xxx)

        /* inputs (LR):

            validpointnc*
            tracer state 5
            tracer state 4
```

```
                tracer state 3
                tracer state 2
                tracer state 1
                tracer state 0
            */
        )
    )
```

**Listing 8. Modgen PDL file for register select PLA used in feature extractor**

```
(module
    /* tracer post processor PLA

    generates enables for different computational blocks.
    */
    (name pla3 )
    (type fsm )
    (in 6 )
    (out 11 )
    (numclocked 0)
    (minterm 14 )
    (output-plane

        (array 11000000000)
        (array 10000000000)
        (array 01000000000)
        (array 00100000000)
        (array 00110000000)
        (array 00010000000)
        (array 00001000000)
        (array 00001000000)
        (array 00000100000)
        (array 00000010000)
        (array 00000001000)
        (array 00000000100)
        (array 00000000010)
        (array 00000000001)

        /* outputs (left to right):

            xstart
            ystart
            xmax
            ymax
            perimeter
            odd perimeter
            xmin,ymin
            xbar
            ybar
            area
            input data

        */
    )
    (input-plane

        (array x10xxx)
        (array 0x0xx0)
        (array 0x0x0x)
        (array 000xx1)
        (array xx1000)
        (array 000x1x)
```

```
            (array 100xxx)
            (array xx1001)
            (array xx1010)
            (array xx1011)
            (array xx1100)
            (array xx1101)
            (array xx1110)
            (array xx1111)

            /* outputs (left to right):

                  selstart*
                  reset
                  override
                  a2
                  a1
                  a0
            */
      )
)
```

**Listing 9. State Transition Table for Contour Tracer**

| current state | | current pixel off | | | | | current pixel on | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | next state | | dX | dY | out | next state | | dX | dY | out |
| high | low | high | low | | | | high | low | | | |
| 0 | 0 | 0 | 7 | 0 | 1 | 3 | 0 | 2 | 1 | -1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 3 | 1 | 4 | -1 | -1 | 1 |
| 0 | 2 | 0 | 1 | 0 | 0 | 3 | 0 | 1 | 0 | 0 | 3 |
| 0 | 3 | 0 | 3 | 1 | 0 | 3 | 4 | 7 | -1 | -1 | 3 |
| 0 | 4 | 2 | 5 | 0 | 0 | 4 | 4 | 6 | -1 | 1 | -4 |
| 0 | 5 | 0 | 4 | 0 | -1 | 3 | 5 | 0 | 1 | 1 | -3 |
| 0 | 6 | 0 | 5 | -1 | 0 | 3 | 6 | 0 | 1 | 1 | -2 |
| 0 | 7 | 0 | 6 | -1 | 0 | 3 | 7 | 2 | 1 | -1 | -1 |
| 1 | 0 | 1 | 7 | 0 | 1 | 3 | 0 | 2 | 1 | -1 | -1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 3 | 1 | 4 | -1 | -1 | 0 |
| 1 | 2 | 1 | 1 | 1 | 0 | 3 | 2 | 4 | -1 | -1 | 1 |
| 1 | 3 | 1 | 2 | 1 | 0 | 3 | 3 | 6 | -1 | 1 | 2 |
| 1 | 4 | 1 | 3 | 0 | 0 | 3 | 1 | 3 | 0 | 0 | 3 |
| 1 | 5 | 2 | 5 | 0 | 0 | 4 | 5 | 0 | 1 | 1 | -4 |
| 1 | 6 | 1 | 5 | -1 | 0 | 3 | 6 | 0 | 1 | 1 | -3 |
| 1 | 7 | 1 | 6 | -1 | 0 | 3 | 7 | 2 | 1 | -1 | -2 |
| 2 | 0 | 2 | 7 | 0 | 1 | 3 | 0 | 2 | 1 | -1 | -2 |
| 2 | 1 | 2 | 0 | 0 | 1 | 3 | 1 | 4 | -1 | -1 | -1 |
| 2 | 2 | 2 | 1 | 1 | 0 | 3 | 2 | 4 | -1 | -1 | 0 |
| 2 | 3 | 2 | 2 | 1 | 0 | 3 | 3 | 6 | -1 | 1 | 1 |
| 2 | 4 | 2 | 3 | 0 | 0 | 3 | 2 | 3 | 0 | 0 | 3 |
| 2 | 5 | 2 | 5 | 0 | 0 | 4 | 2 | 5 | 0 | 0 | 4 |
| 2 | 6 | 2 | 5 | 0 | 0 | 4 | 6 | 0 | 1 | 1 | -4 |
| 2 | 7 | 2 | 6 | -1 | 0 | 3 | 7 | 2 | 1 | -1 | -3 |
| 3 | 0 | 3 | 7 | 0 | 1 | 3 | 0 | 2 | 1 | -1 | -3 |
| 3 | 1 | 3 | 0 | 0 | 1 | 3 | 1 | 4 | -1 | -1 | -2 |
| 3 | 2 | 3 | 1 | 1 | 0 | 3 | 2 | 4 | -1 | -1 | -1 |
| 3 | 3 | 3 | 2 | 1 | 0 | 3 | 3 | 6 | -1 | 1 | 0 |
| 3 | 4 | 3 | 3 | 0 | -1 | 3 | 4 | 6 | -1 | 1 | 1 |
| 3 | 5 | 3 | 4 | 0 | -1 | 3 | 5 | 0 | 1 | 1 | 2 |
| 3 | 6 | 3 | 5 | 0 | 0 | 3 | 3 | 5 | 0 | 0 | 3 |
| 3 | 7 | 2 | 5 | 0 | 0 | 4 | 7 | 2 | 1 | -1 | -4 |

**Listing 9. State Transition Table for Contour Tracer**

| current state | | current pixel off | | | | | current pixel on | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | next state | | dX | dY | out | next state | | dX | dY | out |
| high | low | high | low | | | | high | low | | | |
| 4 | 0 | 2 | 5 | 0 | 0 | 4 | 0 | 2 | 1 | -1 | -4 |
| 4 | 1 | 4 | 0 | 0 | 1 | 3 | 1 | 4 | -1 | -1 | -3 |
| 4 | 2 | 4 | 1 | 1 | 0 | 3 | 2 | 4 | -1 | -1 | -2 |
| 4 | 3 | 4 | 2 | 1 | 0 | 3 | 3 | 6 | -1 | 1 | -1 |
| 4 | 4 | 4 | 3 | 0 | -1 | 3 | 4 | 6 | -1 | 1 | 0 |
| 4 | 5 | 4 | 4 | 0 | -1 | 3 | 5 | 0 | 1 | 1 | 1 |
| 4 | 6 | 4 | 5 | 0 | 0 | 3 | 4 | 5 | 0 | 0 | 3 |
| 4 | 7 | 6 | 1 | 1 | 0 | 3 | 3 | 6 | -1 | 1 | 3 |
| 5 | 0 | 5 | 7 | 0 | 0 | 3 | 5 | 7 | 0 | 0 | 3 |
| 5 | 1 | 2 | 5 | 0 | 0 | 4 | 1 | 4 | -1 | -1 | -4 |
| 5 | 2 | 5 | 1 | 1 | 0 | 3 | 2 | 4 | -1 | -1 | -3 |
| 5 | 3 | 5 | 2 | 1 | 0 | 3 | 3 | 6 | -1 | 1 | -2 |
| 5 | 4 | 5 | 3 | 0 | -1 | 3 | 4 | 6 | -1 | 1 | -1 |
| 5 | 5 | 5 | 4 | 0 | -1 | 3 | 5 | 0 | 1 | 1 | 0 |
| 5 | 6 | 5 | 5 | -1 | 0 | 3 | 6 | 0 | 1 | 1 | 1 |
| 5 | 7 | 5 | 6 | -1 | 0 | 3 | 7 | 2 | 1 | -1 | 2 |
| 6 | 0 | 6 | 7 | 0 | 0 | 3 | 6 | 7 | 0 | 0 | 3 |
| 6 | 1 | 0 | 1 | 1 | 0 | 3 | 2 | 4 | -1 | -1 | 2 |
| 6 | 2 | 2 | 5 | 0 | 0 | 4 | 2 | 4 | -1 | -1 | -4 |
| 6 | 3 | 6 | 2 | 1 | 0 | 3 | 3 | 6 | -1 | 1 | -3 |
| 6 | 4 | 6 | 3 | 0 | -1 | 3 | 4 | 6 | -1 | 1 | -2 |
| 6 | 5 | 6 | 4 | 0 | -1 | 3 | 5 | 0 | 1 | 1 | -1 |
| 6 | 6 | 6 | 5 | -1 | 0 | 3 | 6 | 0 | 1 | 1 | 0 |
| 6 | 7 | 6 | 6 | -1 | 0 | 3 | 7 | 2 | 1 | -1 | 1 |
| 7 | 0 | 7 | 7 | 0 | 1 | 3 | 0 | 2 | 1 | -1 | 1 |
| 7 | 1 | 7 | 0 | 0 | 1 | 3 | 1 | 4 | -1 | -1 | 2 |
| 7 | 2 | 7 | 1 | 0 | 0 | 3 | 7 | 1 | 0 | 0 | 3 |
| 7 | 3 | 2 | 5 | 0 | 0 | 4 | 3 | 6 | -1 | 1 | -4 |
| 7 | 4 | 7 | 3 | 0 | -1 | 3 | 4 | 6 | -1 | 1 | -3 |
| 7 | 5 | 7 | 4 | 0 | -1 | 3 | 5 | 0 | 1 | 1 | -2 |
| 7 | 6 | 7 | 5 | -1 | 0 | 3 | 6 | 0 | 1 | 1 | -1 |
| 7 | 7 | 7 | 6 | -1 | 0 | 3 | 7 | 2 | 1 | -1 | 0 |

**Listing 10. Impulse Response Description for linear Convolver**

```
* impulses responses for image convolver chip
*
* 8/8/84
*
* legal constant values
*
1 0  invert/ true
1 0  done/ not done
*coeff  code    value
G
1    00001    1
1/2   10000    0.5
3/4   01000    0.75
7/8   00100    0.875
15/16 00010    0.9375
C
0    11111x0  0
1    01111x1  1.0
1/2   10111x1  0.5
1/4   11011x1  0.25
1/8   11101x1  0.125
1/16  11110x1  0.0625
E
*
* Impulse responses to put on chip
*
*
* LPF #1
*
G=7/8
1/8 1/8 1/8
1/8 1/8 1/8
1/8 1/8 1/8
*
* LPF #2
*
G=3/4
1/8 1/8 1/8
1/8 1/4 1/8
1/8 1/8 1/8
*
* LPF #3
*
1/16 1/8 1/16
1/8  1/4 1/8
1/16 1/8 1/16
*
* LAPLACIAN #1
*
-1/8 -1/8 -1/8
-1/8  1   -1/8
-1/8 -1/8 -1/8
```

```
*
* LAPLACIAN #2
*
 0  -1/4  0
-1/4  1   -1/4
 0  -1/4  0
*
* LAPLACIAN #3
*
 1/8 -1/4  1/8
-1/4  1/2 -1/4
 1/8 -1/4  1/8
*
* SOBEL X
*
-1/4  0  1/4
-1/2  0  1/2
-1/4  0  1/4
*
* SOBEL Y
*
 1/4  1/2  1/4
  0   0   0
-1/4 -1/2 -1/4
*
* EDGE ENHANCEMENT
*
-1/16 -1/16 -1/16
-1/16  1   -1/16
-1/16 -1/16 -1/16
*
* ALL PASS #1
*
0 0 0
0 1 0
0 0 0
*
* ALL PASS #2
*
0 1 0
0 0 0
0 0 0
*
* ALL PASS #3
*
0 0 0
0 0 0
0 1 0
*
* Compass Gradient West
*
G=3/4
1/4  1/4 -1/4
1/4 -1/2 -1/4
```

```
1/4  1/4 -1/4
*
* Compass Gradient East
*
G=3/4
-1/4  1/4 1/4
-1/4 -1/2 1/4
-1/4  1/4 1/4
*
* Compass Gradient North
*
G=3/4
1/4  1/4  1/4
1/4 -1/2  1/4
-1/4 -1/4 -1/4
*
* Compass Gradient South
*
G=3/4
-1/4 -1/4 -1/4
1/4  -1/2  1/4
1/4   1/4  1/4
E
```

**Listing 11. Data-Path Description for Non-Linear Low-Pass Filter**

```
*
  non linear low pass filter processor
  5/2/85
*

LEFT 'laplac_in' 'allpass_in'  * input signals *
RIGHT 'out' 'thresh'           * output signal and threshold *

ORGANIZATION
        CNT       NODATA
        MSB       DATA
        EVEN      DATA
        ODD       DATA
        EVEN      DATA
        ODD       DATA
        EVEN      DATA
        ODD       DATA
        EVEN      DATA
        ODD       DATA
        EVEN      DATA
        GND       NODATA

SLICE

        'allpass_in' > delay > 'allpass'

        * generate low pass signal *

        'laplac_in' > minus1 > delay > adder ('inb'='allpass') > satregister > 'lowpass'
        'allpass' > delay > 'allpass1'

        * generate decision *

        'laplac_in' > absvalue > delay > adder ('inb'='thresh')> delay

        * generate output *

         'allpass1' > mux2to1('inb'='lowpass') >  delay > modify > 'out'
```

# APPENDIX E USING THE IMAGE PROCESSING SYSTEM

To aid in the development and characterization of the image-processing/recognition system, the custom hardware was interfaced to a SUN workstation. The image processing hardware can be controlled via a graphical program, "ittool". Ittool allows the user to set the system configuration, gather results from the chips and display these results.

## E.1 THE GRAPHICAL INTERFACE

The program has three windows (figures 1-6), the multi-use graphics window at the bottom of the tool, the control window in the middle of the tool and a small message window at the top of the tool. The message window is used to report errors and to display general information. The control window is used to manipulate some of the chips (top line), to control the writing of hardware control registers and the frame buffer ($2^{nd}$ line), to control the digitizer and frame buffer operation ($3^{rd}$ line), to control the contour tracer ($4^{th}$ line), and to select the display mode of the graphics window ($5^{th}$ line), and to select the features used by the recognizer. Five types of information can be displayed in the graphics window, including: a graphical representation of the most of the hardware system, the contours of objects that have been traced, the histogram of the image in the frame buffer, the symbolic representation of objects after recognition, the curvature and slope signals, and the impulse responses of the two linear convolvers.

### E.1.1 The Control Window

The control window contains "buttons" spread over 5 lines. "Pressing" the "button" (or selecting a menu item) causes some action to occur. Buttons on the first (top) line control the input ROM, output ROM, logical convolver and the sorting filter. For each ROM, there are two modes of operation: active and inactive. Selecting the proper button toggles the mode of the ROM. There are 4 modes of operation for the logical convolver: bloat 0 (inactive), bloat by 1, bloat by 2, and bloat by 3. These options can be

selected by the "pressing" the bloat button. The type of operation performed by the sorting filter can be chosen with the $4^{th}$ button on the top line. The region of operation of the sorting filter is controlled by the $5^{th}$ and $6^{th}$ buttons. The button marked "H" controls the horizontal region of operation while the button marked "V" controls the vertical region of operation. Selecting "0" makes the region of operation a single point. Selecting "1" or "2" makes the region 2 points and selecting "3" makes the region 3 points.

The second line in the control window gives the user the ability to write into the hardware registers on the Imaging Technology boards and the recognition board, to change or store the hardware configuration and to load or store the frame buffer. The first button controls which register is changed. For all choices except "Set-up" and "Frame Bu" when a new value is entered and the user presses <return>, the register is loaded with the new value. When "Set-up" or "Frame Bu" is selected, the new value entered should be a file name. Pressing the read button will read data from the file and pressing the write button will write data to the file. If "Frame Bu" is selected, the green frame buffer will either be loaded from the file or written to the file. If "Set-Up" is selected, the current hardware configuration will either be loaded from the file or written to the file. For the "Set-Up" selection, the actual file name used is the concatenation of ".itinfo." and the name entered by the user.

The third line of buttons makes it possible to change the way in which video signals are digitized. Selecting the first button, toggles the digitizer between 10 MHz and 5 MHz modes (512 pixel lines and 256 pixel lines). Selecting the second button toggles the number of lines digitized. The third button indicates whether both interlaced fields are displayed. The "Red", "Green" and "Blue" buttons indicate if "Watch", "Grab" and "Clear" will affect the corresponding frame buffers. When "Watch" is pressed, the digitizer will continuously digitize all selected colors. When "Grab" is pressed the data in the selected colors will be frozen in the corresponding frame buffer. "Clear" will set the selected frame buffers to all zeros.

The fourth line of buttons is used to control the contour tracer. The mode button (third button) indicates the tracing mode. When "Man" (manual) is selected, tracing stops after every encountered contour. When "Auto-I" (auto image) is selected, tracing will stop after a valid image is found. Tracing will stop only when the end of a frame is found when "Auto-F" (auto frame) is selected. Pressing the

"Acquire" button commands the tracer to load a new frame into its buffer and trace according to the selected mode. The "Next" button can be used to have the tracer search for the next object. If "read" is selected, contour data is taken not from the tracer but from a file (the name is the same as that used for changing the hardware set up). If "write" is selected, the traced data is written a file. These options are used to save template objects and recall them later.

The first two buttons on the fifth line control the display modes. The first button indicates what is being displayed in the graphics window. The second button indicates whether diagnostic information should be printed when contour tracing. The next four buttons indicate which features will be used by the recognizer. The next button (mse n) sets the the length of one FIR filter used in the feature computation. Pressing "train" causes a new "templates" file to be generated using information stored in the file, "templates.input". Pressing "equal" causes the image in the buffer to be histogram equalized (the histogram should be computed first).

### E.1.2 Graphics Window

In the schematic mode, the graphics window shows a graphical representation (figure 1) of part of the system. The configuration can be changed by moving the cursor to the appropriate area (a box will appear when something can be changed) and pressing the left or center buttons.

When changing the impulse response of the convolvers, the available responses (figure 2) will be displayed in the window. When one of the impulse responses is selected, the schematic will be displayed and the change will be made.

If the "hist" option is selected in the control window, the histogram and other information regarding the image in the frame buffer will be displayed (figure 3).

If the "contour" option is selected, the contours of any objects that have been traced will be displayed in the graphics window (figure 4). In addition, the values of the features for each object and the results of the recognition will be displayed. The two top choices and the corresponding errors are printed.

When "recog" is chosen, the continuous recognition mode will be activated by pressing "Acquire".

In this mode, frames will be continuously processed and the images recognized. The results of the recognition are shown symbolically in the window (figure 5).

The filtered curvature and slope of the last traced object are displayed when "curvat" is selected (figure 6).

### E.2 Input and Output Formats

There are several files that are read by ittool: the set-up file, the template file, "templates", the template input file, "templates.input", the curvature files and image raster files.

The set-up files are not human readable and are simply a binary dump of a structure.

The curvature files written and read by the program have the same format. The first line contains (xmin, xmax, ymin, ymax, perimeter, even perimeter, area $\frac{A}{P^2}$, true perimeter) for the object. On the following lines, the value of the curvature at each point on the contour is given and there will be "perimeter" curvature values.

The template input file has one line for each object being trained. The format for each line is the name of the object followed by the number of images at each particular size. For example, one line might be "square 4 4 4 4 4". This means that there is a file called "square.image" which contains the curvature for 20 images. There are 4 images at each of 5 different sizes. The format of the curvature file is the same as that described above. In the template file, for each object, the object name is given and the name of the pixrect file for that object (for symbolic representation). Following this are the area, perimeter and $\frac{A}{P^2}$ for the standard object size. Next, the area, $\frac{A}{P^2}$,$P\,\sigma^2_{curv}$ and $\sum_{curv>0} curv$ is given for each area. For example, for the square, part of the template file would be:

```
square square.pr 645 103 0.060065
    271 0.062143 150 4
    756 0.060389 248 41
    2162 0.058840 333 113
    6431 0.058582 406 132
```

The raster files have an 8-byte header. The first four bytes are the number of columns (lsb first) in the image. The second four bytes are the number of rows (lsb first) in the image. After the header, the

raster scanned image (row by row, left to right) is stored with one pixel per byte.

To run the program, one can type "ittool >/dev/null &". This will prevent the recognition statistics from being displayed that are generated and dumped each time the continuous recognition mode is chosen. If there is no template file or set-up file in the current directory, the program will still run. Without a template file the program will not attempt to recognize any objects.

**Figure 1. Ittool in "schematic" Mode**

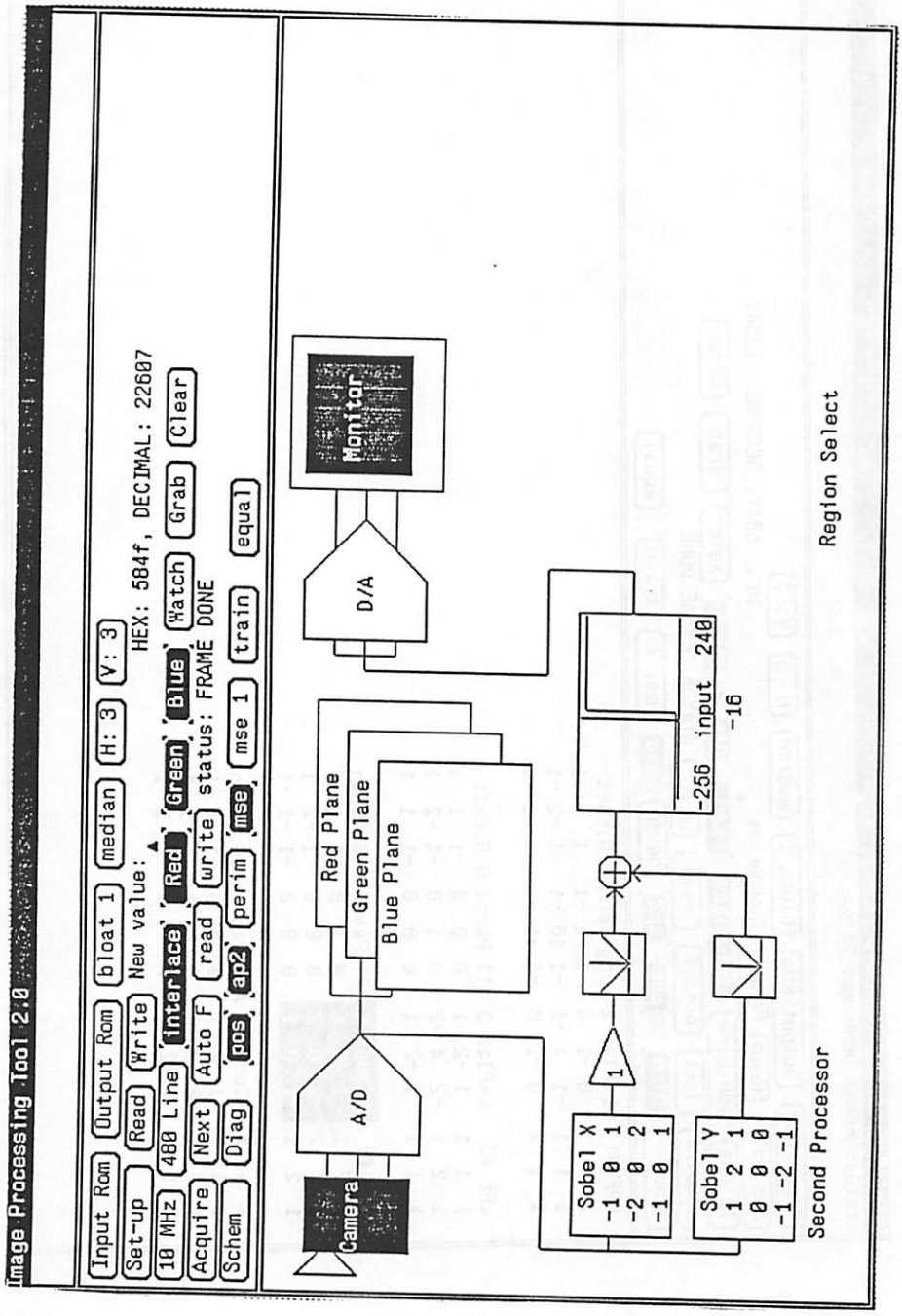


**Figure 2. Ittool in "impulse response" Mode**

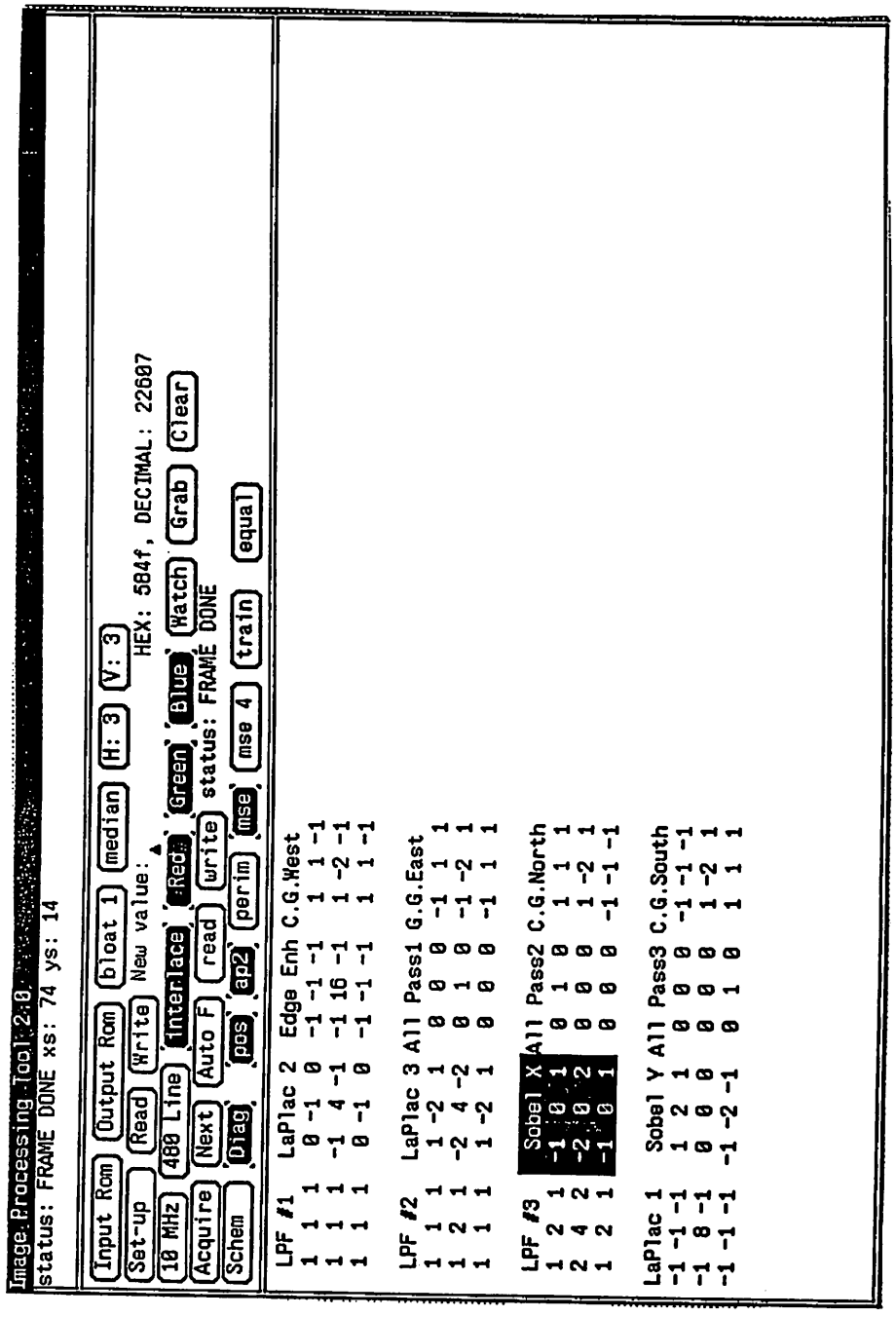


**Figure 3. Ittool in "histogram" Mode**

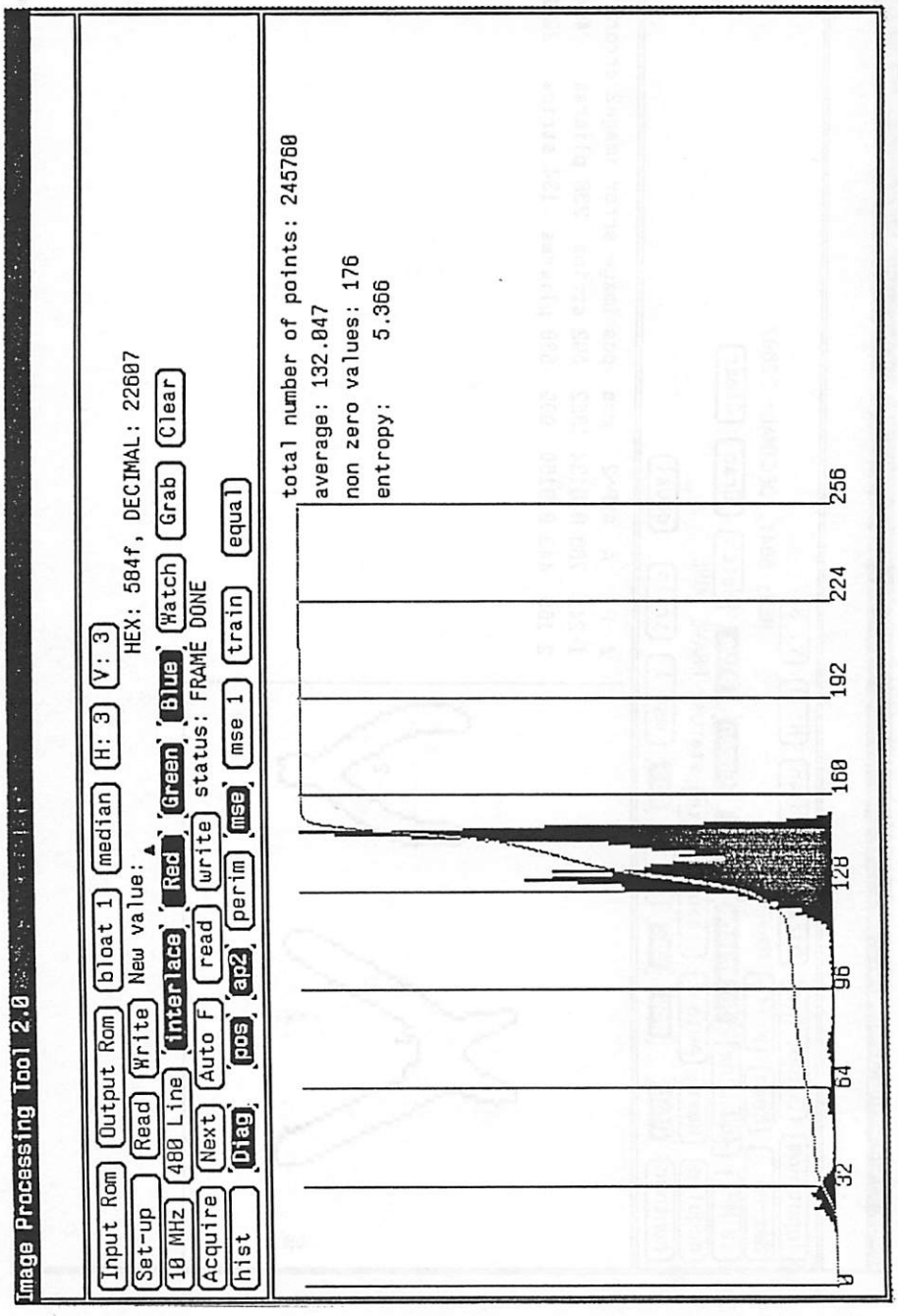


**Figure 4. Ittool in "contour" Mode**

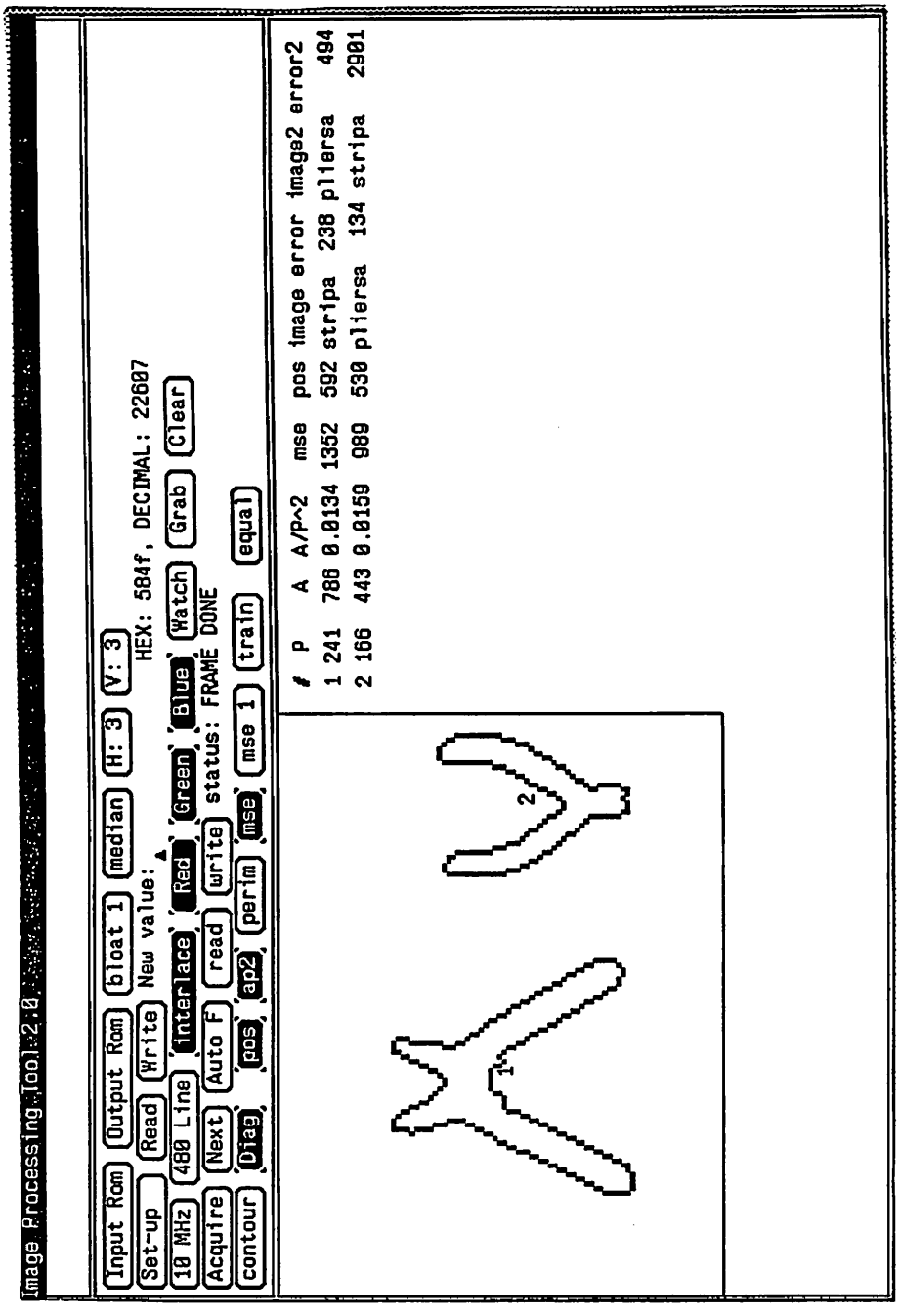


**Figure 5. Ittool in "recognition" Mode**

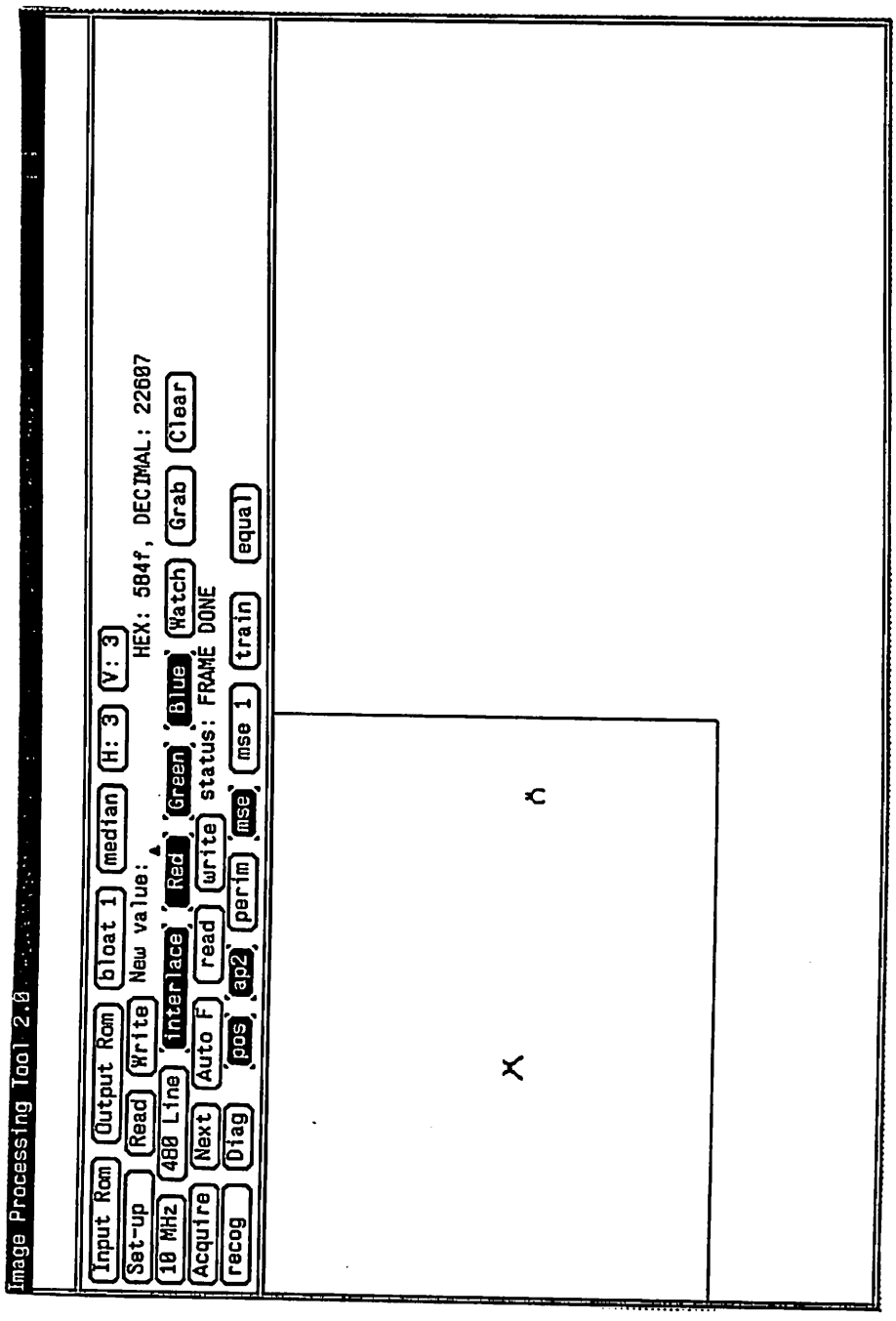
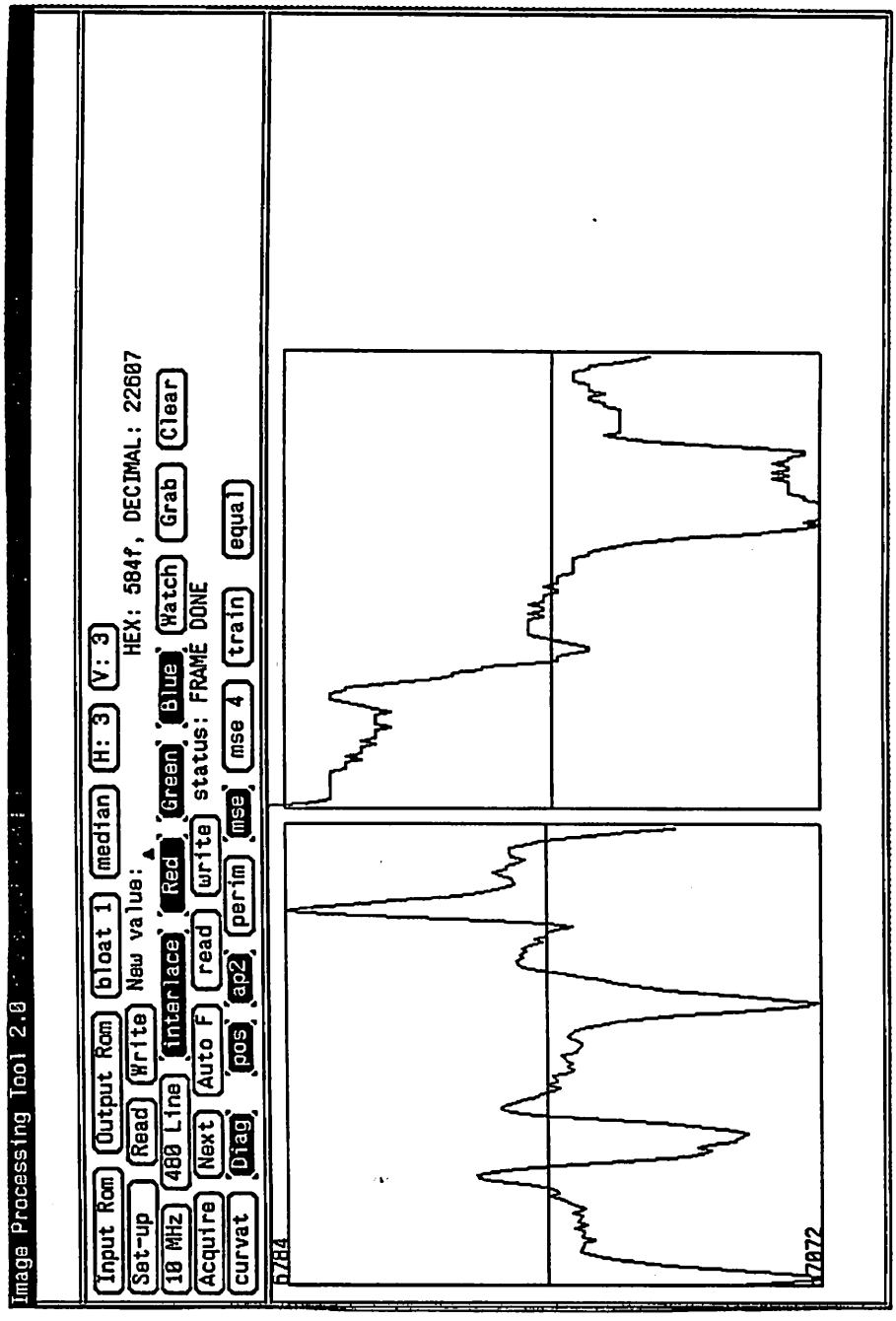


**Figure 6. Ittool in "curvature" Mode**



# APPENDIX F IMAGE PROCESSING PROGRAMS

A set of programs was written to make simulations of image processing algorithms convenient. A standard input/output scheme was used so that programs could be piped together to cascade different functions. The programs include:

| | |
|---|---|
| conv | performs nxn convolution |
| add | adds two images together |
| bloat | performs 3x3 expansion |
| bloatbuff | performs 3x3 expansion on data in green buffer |
| combine | combines up to four images for display at one time |
| dither | dithers images |
| ditherprint | dithers an image and prints it on the laser printer |
| downsample | downsamples an image |
| expand | expands each point in an image into a rectangle |
| hist | computes the histogram of an image |
| histequal | performs histogram equalization |
| invert | inverts the green frame buffer |
| lut | performs arbitrary point-wise modifications |
| noise | adds different types of noise to an image |
| offsetgain | adds an offset to an image an multiplies by a constant |
| putsunraster | converts an image from the format for the programs to SUN format |
| rcdisp | display an image on IT color display (green) |
| rdisp | display an image of the SUN color monitor or BW window |
| read | read image from IT green frame buffer |
| readsunraster | converts images in SUN raster format to the format used in these programs |
| thresh | thresholds an image |

Programs with a single image input, typically read the input from standard in. Programs with a single image output, typically write this output to standard out.