ADAP: A Mixed-Signal Array Processor With Early Vision Applications

by

David Andrew Martin

Bachelor of Science
Electrical Engineering and Computer Science
Massachusetts Institute of Technology (1991)

Master of Science
Electrical Engineering and Computer Science
Massachusetts Institute of Technology (1991)

Submitted to the Department of
Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

August, 1996

Signature of Author ______________________________
Department of Electrical Engineering and Computer Science
August, 1996

Certified by ______________________________
Hae-Seung Lee
Professor, Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ______________________________
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Students

ADAP: A Mixed-Signal Array Processor With Early Vision Applications

by

David Andrew Martin

Submitted to the Department of
Electrical Engineering and Computer Science
On August 7, 1996 in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Abstract

Many low level vision algorithms require only low to moderate precision (6 to 8 bits). For these applications, a special purpose analog circuit is often a smaller, faster, and lower power solution than a general purpose digital processor. However, because these analog chips are only suitable for one very specific function, they are often expensive, low volume products. This thesis presents a more general purpose programmable mixed-signal array processor, ADAP, that combines the flexibility of a digital processor with the small area and low power of an analog circuit. It achieves a processing efficiency in terms of power and area superior to that of a comparable digital processor. Each processor in the array has a digital control unit, an analog switch fabric, an analog storage unit, and an analog arithmetic unit with an accuracy of 7 bits. The analog arithmetic unit utilizes a unique circuit that combines a cyclic switched capacitor A/D and D/A to perform addition, subtraction, multiplication, and division. Each processor cell performs, in parallel with all of the other processor cells, 0.8 million operations per second, consumes 1.825mW of power, and uses 700μm by 270μm of silicon area. The chip was fabricated in Hewlett Packard's 0.8μm triple metal CMOS process. An array of these processors was used to successfully perform an edge detection algorithm, and a sub-pixel resolution algorithm executed on the array was able to increase the resolution of the edge locations by a factor of four.

Thesis Supervisor: Professor Hae-Seung Lee

Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

When I leave MIT, it will be exactly ten years since I first arrived as a freshman. I have learned so much at MIT, but I am looking forward to learning even more in the world outside.

I must first of all thank my thesis supervisor, Professor Hae-Seung Lee. Professor Lee's advice and insight into analog circuits was invaluable. He also kept my research headed in the right direction and had the right attitude about meetings. It has been a great pleasure and a wonderful learning experience to work with him. Dr. Ichiro Masaki also provided an enormous amount of supervision for my work; he helped me understand the system issues that drove my circuit design; thanks to him, I was able to see the big picture. Dr. Masaki was also instrumental in getting the entire project started and did a lot of work behind the scenes dealing with funding.

Professor Charles Sodini also served on my thesis committee and provided numerous helpful suggestions for both my thesis proposal and the thesis itself. Jason Bergendahl gave me lots of information about how my circuits could be used in a real system, and he proofread my thesis. Professor Berthold Horn and Professor John Wyatt gave me various valuable pieces of advice on vision and circuit theory; Professor Wyatt also did a lot of work overseeing the MIT Analog Vision Chip Group. Sam Reynolds at MOSIS provided enormous amounts of assistance cutting the red-tape involved in fabricating a chip.

I'd like to thank Charles Miller and Professor Alan Oppenheim for being incredible academic advisors. Dr. David Clark and Dr. David Tennenhouse were instrumental in guiding my research for my Bachelor's and Master's degrees. Many thanks to Professor James Roberge, Professor Oppenheim, and Dr. Tennenhouse for being on my Area Exam Committee at the last minute. Professor Roberge was one of the inspired teachers at MIT who taught me that analog circuits could be fun. I have also realized how much I learned from my co-workers at Motorola during my 6A jobs. Craig Holt, Phil Giangarra, Spencer Hu, and Eddie were especially helpful.

I'd also like to thank Marilyn Pierce, Monica Bell, Lisa Bella, Beth Chung, Kurt Broderick, and Carolyn Zaccaria for helping me traverse the administrative labyrinth of the MIT EECS Department. Also, many thanks to the staff of the instrument room in Building 38 for supplying me with chips, instruments, and overhead projectors over the years.

The students at MTL provided huge amounts of both friendship and technical advice. Special thanks goes to my officemates over the years. Andy Karanicolas gave me tons of help with analog circuit design, circuit testing advice, and proofread parts of my thesis. He also provided hours of excellent political arguments, which probably lengthened both of our graduate careers. Jungwook Yang gave me lots of help with Cadence as well as sharing some of his wife's excellent cooking with me. Jen Lloyd helped me with Cadence, talked about the stock market and the Web in general a good deal, and took endless phone messages for me. Myron Freeman (Fletch) and Jeff Gealow were instrumental in keeping the computers and CAD programs in the circuit part of MTL running. Joe Lutsky, Steve Decker, Chris

Umminger, Ig McQuirk, Vincent Peiris, and Paul Yu also provided random bits of circuit advice and general friendship. Kush Gulati and Aiman Shabra went to my practice talks and proofread parts of my thesis.

I'd like to express my gratitude to all of my friends from outside of work. Thanks to all of my apartment-mates, especially John Miller, Eric Emerson, Ernie Cavicchi, Peter Panic, Keith Reilly, Rick Umali, Tamara Cadd, and Tammy Sue Roorda, over the years who helped make 30 Adrian Street a pleasant place to live. Thanks to my undergraduate friends Alex Cano-Ruiz, Andre DeHon, Andrea Zimmerman, Anne Francomano, Barbara Brady, BJ Davis, J. Michael Hammond, David Harris, David Plass, Debbie Kuchnir, Farzana Khatri, Grace Chen, Karyn Baum, Lara Garrett, Luis Rodriguez, Malini Nadig, Marcie Black, Pablo Munguia, Pratima Rangarajan, Rob French, Rukiye Devres, Sam Niles-Peretz, Sheela Magge, Shella Farooki, Shelly Friedland, Tolga Erdogus, Theresa Cheng, and Tracey Adams. Their friendship has been a source of happiness, and their romantic entanglements have been a source of amusement. Thanks to Heidi Warriner for everything. My freshman roommates Harry Sverdlove and Richard Cheng deserve special mention for always telling me when I was doing something stupid. Thanks also to my LP friends Emily Wallis, Vernon Imrich, Raymie Stata, Jeff Schachter, Jason Solinsky, and Suzie Blevins. Thanks to all of my sailing buddies Dwight Brown, Katie Joynt, Kate Kelly, Magda, Sue and Steve Ostrowski, Peter, and Brian, for giving me something fun to do that had nothing to do with engineering. Thanks to Hatch Brown and Fran Charles for a wonderful job of running the MIT Sailing Pavilion. Finally, special thanks to my best friend from high school, Steve, for being sympathetic to all of my complaints for the past 15 years.

This research was supported by the National Science Foundation under contract 9117724-MIP. The chips were fabricated at Hewlett Packard through MOSIS, and the PCBs were fabricated at Multek, also through MOSIS.

Dedication

The ten years that I have spent at MIT would not have been possible without the continual love and support of my family. It was not until I left home that I realized just how important they were. This thesis is dedicated to Mom, Dad, and my sister Anne.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As digital processors continue to increase in speed due to continuing advances in CMOS processing and digital circuit design techniques, many electronic systems have become almost entirely digital, having only an A/D and a D/A at the edges. For many applications, this is the optimal approach since digital circuits offer programmability, high speeds, numerous automated design methodologies, and the ability to easily scale with process advances. However, there are some applications for which an analog or mixed-signal system offers superior performance and more efficient use of power and silicon area. One such application is early vision processing; a number of analog vision chips which are faster, smaller, and lower in power than digital image processors have been developed [1, 2, 3, 4, 5, 6]. The primary limitation of these chips is that they are all specific to one application; a digital solution might offer less performance, but it would be far more flexible and programmable. The goal of this thesis is to design a mixed-signal processor that combines the high performance of a special-purpose analog chip with some of the programmability and flexibility of a digital system. The key idea is that although the mixed-signal processor will not be as flexible or programmable as a digital processor, it will be flexible enough to perform any operation necessary for a certain class of algorithms while at the same time preserving most of the low area, low power, and high speed of a special purpose analog chip.

Vision processing provides an excellent example of a class of algorithms appropriate for this sort of processor. Since most vision algorithms are parallel in nature, they are well suited to parallel processor architectures. Therefore, a processor that uses silicon area and power very efficiently is very important. In addition, many vision algorithms require a processor with only 7 to 8 bits of accuracy, so the limited resolution of a mixed-signal processor is not a problem. A test application requiring a processor with a high degree of parallelism but only 7 to 8 bits of accuracy was chosen. This application is a stereo vision processor for an intelligent cruise control (ICC) system. An ICC system detects the distance to the car ahead and adjusts the motor speed to keep this distance constant, as shown in Figure 1.1.

The processor designed for this application is an array processor chip. It consists of a rectangular array of processors, each of which stores and processes its data in analog form but is controlled by digital signals. This will provide an opportunity to see how much generality can be introduced into an analog system, to explore the performance tradeoffs involved, and to compare the tradeoffs with those of a digital processor.

The ALU of the processor uses a special circuit that combines an A/D with a D/A. The processor will therefore be called the ADAP (for A/D, D/A Processor) chip.

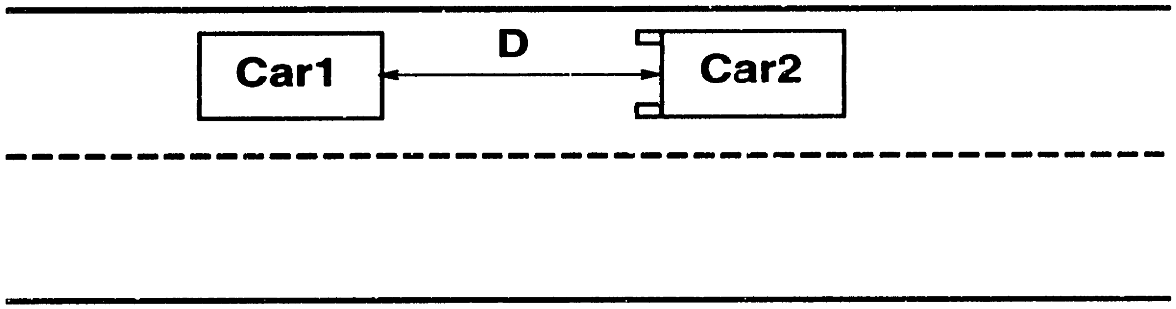


Figure 1.1: Intelligent Cruise Control: Car2 adjusts its speed to keep *D* constant.

1.1 Past and Related Work

1.1.1 Vision for Automotive Control

There are numerous projects underway at major car companies to give an automobile the ability to sense objects ahead and either respond to them automatically or notify the driver of them. Some projects, such as PROMETHEUS, a European research project, have attempted to totally automate the car so that the car can drive itself [7, 8] on public roads. These vehicles include vision sensors and motion sensors along with numerous control units that detect other vehicles, lane markings, and traffic signs. Other projects, such as the Canadian FAGV, are designed to operate in better defined industrial sites [9].

Some of these projects use radar to detect objects. Delco Electronics is working on a system that uses radar to warn the driver of objects that are too close [10]. Raytheon has developed a millimeter wave radar system for intelligent cruise control systems which can track more than one object at a time [11]. Toyota is working on a radar collision avoidance system [12]. Ford is also working on a radar system. The initial projects are simply advanced collision detection systems, but eventually, they will evolve into intelligent cruise control systems [13]. One problem with these methods is that they can only detect objects. They are not able to detect line markings on the road. Also, radar equipment can be very expensive [13]. An additional cost is that any system emitting radar signals will need to be regulated by the government. This regulation will add another set of costs. On the other hand, a vision based system will tend to have the same strengths and weaknesses as human vision. The same environmental conditions which make it difficult for a human to see the road ahead would be the same conditions that degrade the operation of a vision based system. Thus, the average driver, who might not understand how radar systems operate, will still have a good understanding of when a vision based system will be able to operate.

For this reason, there are numerous projects aimed at using vision systems for automotive applications.

Several systems which use lane markings as well as object detection to guide the vehicle have been developed [14] [15]. One of these systems uses an algorithm that searches for symmetric objects in order to find a car in a camera image [16].

Several systems have been developed that use stereo vision for detecting depth. Toyota has developed an experimental system that uses 64 digital processors in parallel for a stereo algorithm [17] subsystem. CLEOPATRA, a European consortium, has developed a vision based ICC system that includes a stereo vision system intended primarily as a warning system. The system uses several high speed DSP chips for processing [18]. One stereo system has been developed that uses linear cameras (only one long row of pixels, not a square matrix of pixels) to reduce the processing time. This system uses two high-speed DSP chips for processing [19].

Finally, some experimental systems try to use several methods at the same time. Hyundai has a system, PRV II, which uses GPS, dead reckoning, image processing, and several types of processors to determine its location and to detect other cars [20].

Many of these vision-based systems use some variation of the standard stereo vision algorithm when actually computing the distance to an object. Some work has been done optimizing various stereo algorithms and camera calibration methods for this application [21].

All of these systems require large racks of electronic equipment including power supplies, computers, and special I/O boards. Most of the research work has concentrated on the initial steps of developing vision algorithms, sensors, calibration issues, and software, not on developing actual low-cost, low-power systems.

GM has developed an experimental intelligent cruise control system which uses a simple stereo algorithm to determine distance [22]. It used a large amount of expensive, power-hungry electronic equipment. ADAP can implement a comparable system at a reasonable cost in terms of power consumption and silicon area.

1.1.2 Array Processors

Most digital array processors are intended for specific purposes. These include neural net chips [23], [24]; data compression chips [25], [26]; and filters [27]. All of these are designed to perform only one task, but perform it much faster and with less hardware than a general purpose SIMD computer. There are a few digital arrays which can perform more than one task. One of these, IMAP [28], is a SIMD processor intended for, but not limited to, image processing applications.

Most analog arrays are even more application-specific. A number of analog vision chips have been designed which perform specific vision tasks such as motion detection [29], finding the focus of expansion [30], and image smoothing [3]. Many slightly more general analog neural nets have been designed in which the function of each synapse and the interconnection between synapses cannot be changed [31, 32, 33]. Other analog neural nets have been designed for which the connections between synapses is programmable [34, 35, 36, 37].

One analog processor intended for vision applications, called APAP, has been designed in which the connections between cells and certain multiplicative constants for the cells are programmable, but each cell performs the same sum-of-weighted-inputs function [38]. None of these analog arrays have programmable processors *and* connections.

Several analog chips have been developed that implement stereo algorithms. In [39], a chip which performs the entire stereo algorithm on one chip is described. The system is very compact (on only one chip) and low power, but many algorithmic details, such as the maximum disparity and correlation metric, are fixed. In addition, it can only use 1-D edge filters, not the more robust 2-D edge filters. Another chip designed for stereo applications is described in [40]. This imager chip uses special filtering circuitry on the imager chip itself to preprocess the image data for the rest of the stereo system. This saves power and cost by eliminating processing overhead, but the filter functionality is fixed and cannot be modified.

1.2 Different Processor Designs

There are many ways in which to design the processor. It can be digital or analog, serial or parallel, pipelined or not, and optimized for various performance characteristics. This section will consider a few very broad categories of processor design.

1.2.1 Types of Processor Arrays

Three classes of processors are Single Instruction, Single Data (SISD), Single Instruction, Multiple Data (SIMD), and Multiple Instruction, Multiple Data (MIMD). A SISD processor can only perform one instruction on one piece of data at a time. It is usually too slow for many vision algorithms because of the enormous input data rates. A SIMD processor can perform the same instruction on many pieces of data at the same time; it can handle the higher data rate but often cannot handle the large number of calculations (both integer and floating point) required for vision processing. A MIMD processor can perform different instructions on multiple pieces of data at the same time. Therefore, for vision applications, a MIMD architecture provides the best performance to deal with the large number of inputs and operations characteristic of vision processing.

1.2.2 Digital Computation

A digital processor has the advantage of having as much resolution as the user desires at a cost in area, power, and speed. They can also be programmed to perform many different tasks. Their disadvantage is that for some applications which do not need a high amount of accuracy and do not need to be completely general purpose, the area, speed, and power of a digital processor may be wasted.

Two digital SIMD designs described in [28] and [41] will be used for comparison purposes. These processors are, like ADAP, arrays of processor cells that are intended for image processing applications. They are described in detail in Section 8.2.

1.2.3 Analog Computation

An analog processor has the disadvantage of having limited accuracy, more susceptibility to noise, and, usually, far less programmability. However, for a limited number of functions, it can perform these functions more quickly in far less area with less power. One advantage in area is that it needs only one wire to carry an analog value whereas a digital system needs N wires for N bits of accuracy. Thus, data channels and switch fabrics can be far smaller in analog systems. Also, certain arithmetic operations can be performed, at a limited accuracy, more efficiently in an analog processor.

1.3 Organization of the Thesis

This chapter has introduced the motivation behind ADAP and discussed previous work in vision systems and processor design. Chapter 2 describes the stereo vision algorithm in detail. Chapter 3 describes the high level ADAP architecture and suggests several system configurations for ADAP. Chapter 4 describes the ADAP processor and its constituent circuits in detail. Chapter 5 describes the physical layout of the ADAP processor. Chapter 6 describes the test setup used to verify the performance of the ADAP processor. Chapter 7 presents the experimental results from testing the ADAP chip. Conclusions and suggestions for future work are presented in Chapter 8.

Appendix A analyzes the error sources in the ADAP processor. Appendix B contains detailed timing information. Appendix C provides examples of how to program the ADAP array chip and corresponding C code. Appendix D provides pin and package information for the ADAP array chip. Finally, Appendix E provides instructions for using the ADAP test setup.

Chapter 2

Stereo Vision Algorithm

Although ADAP is intended to be useful for many different types of vision processing tasks, an automotive intelligent cruise control application was chosen for testing purposes. In this system, there are two[1] cameras located a certain distance apart in the horizontal plane. The images from these cameras are passed to a vision system, which determines the distance to the car ahead. The motor speed is then adjusted to keep this distance constant.

This chapter explains the formula used in stereo vision, explains the actual algorithm implemented on the ADAP chip, and describes two techniques used to increase the system's resolution and robustness.

2.1 The Distance from Disparity Formula

The human eye uses several methods to determine the distance of objects in a scene. One of these methods is to use the difference between left and right images. This technique, called binocular stereo, is based on a simple formula [42].

In Figure 2.1, f is the focal length of the cameras, B is the distance between the centers of the two cameras, and Z is the distance from the cameras to the object. X_2 and X_1 represent the x-coordinate of an object's image in the left and right cameras, respectively. By matching similar triangles:

$$\frac{X_2}{f} = \frac{X + B/2}{Z} \tag{2.1}$$

$$\frac{X_1}{f} = \frac{X - B/2}{Z} \tag{2.2}$$

$$\tag{2.3}$$

$$X_2 = \frac{f}{Z}(X + B/2) \tag{2.4}$$

[1]The actual system will have 3 cameras. This is discussed in Section 2.4.

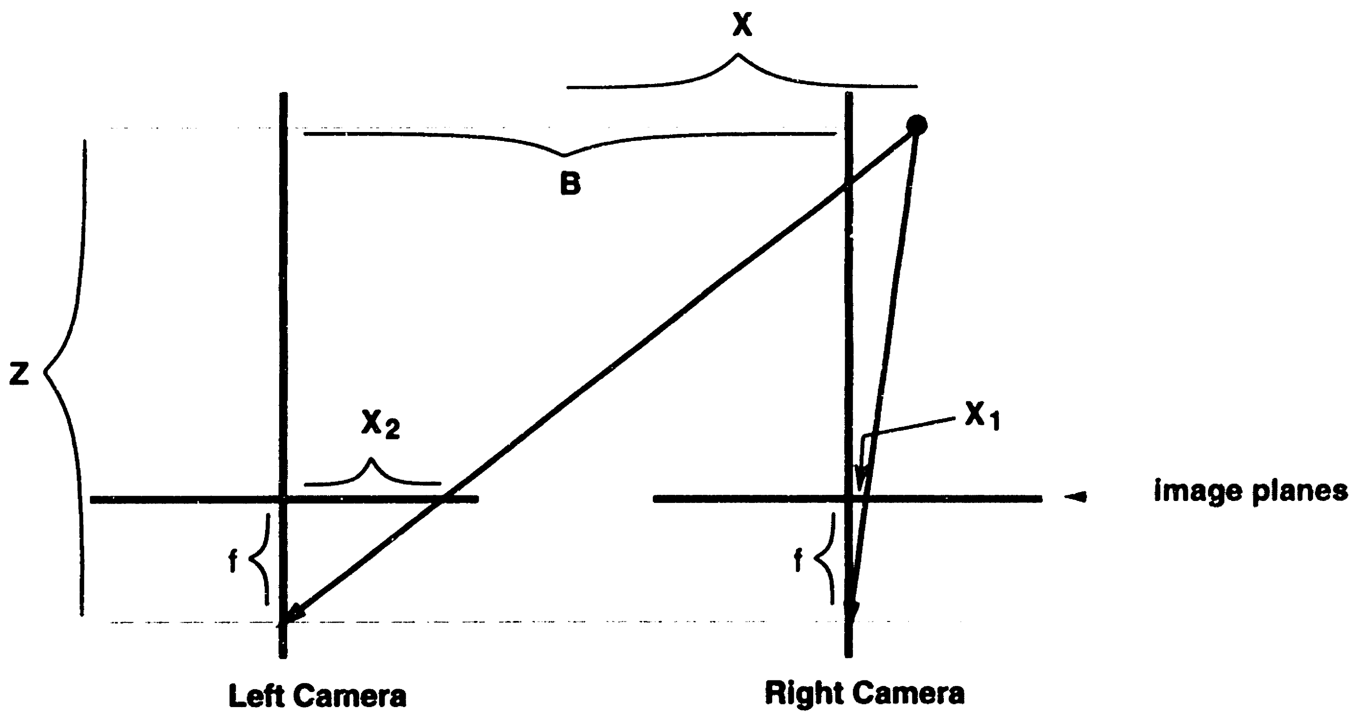


Figure 2.1: Binocular Stereo Example, $D = X_2 - X_1$

$$X_1 = \frac{f}{Z}(X - B/2) \tag{2.5}$$

The algorithm now defines a variable called the disparity(D). The disparity is equal to the difference between the x-coordinate of the left image and the x-coordinate of the right image.

$$\begin{aligned} D &= X_2 - X_1 & (2.6)\\ D &= \frac{f}{Z}(X + B/2 - X + B/2) & (2.7)\\ D &= fB/Z & (2.8) \end{aligned}$$

Finally, rearranging terms yields the equation:

$$Z = fB/D \tag{2.9}$$

Given f and B, all that the stereo-based cruise control system needs to do is to match corresponding points in the left and right images and calculate the disparity D. Putting the disparity into Equation 2.9 provides the distance to the car.

2.2 Edge Detection

The first step of the stereo algorithm is to generate an edge map. Since the cruise control application only requires that the distance to the car be found, the image is converted to a vertical edge map. This will eliminate all information in the image except that associated with vertical edges, reducing the amount of processing that must be done on the image. It is possible to use horizontal edges as well, but that would add processing time. Vertical edges are used instead of horizontal edges because it is easier to eliminate other objects in the scene using vertical edges. A car will usually have stronger vertical edges than any other object in the image. On the other hand, the horizon will have stronger horizontal edges than a car. Finally, the other objects of potential interest to an automotive vision system, such as lane markings, are more easily detected with vertical edges.

In the actual algorithm, two edge maps are actually produced for each image. One is for positive edges, in which the pixel values, or brightness, increase from left to right, and one is for negative edges, in which the opposite occurs. The edge map is generated by replacing the value of each pixel with a scalar number, *Positive_Edge_Strength* or *Negative_Edge_Strength*, representing how much the intensity changes in the neighborhood of that pixel. An example of the way in which this is done is shown in Figure 2.2. This edge detection algorithm is an example of a Sobel filter [43]. For each pixel P, the pixels nearby are multiplied by a weight, K_n. The pixels on the right are then added and divided by the sum of the left pixels using Equation 2.10:

$$Positive\_Edge\_Strength = \frac{\sum right\_pixel_n K_n}{\sum left\_pixel_n K_n} \tag{2.10}$$

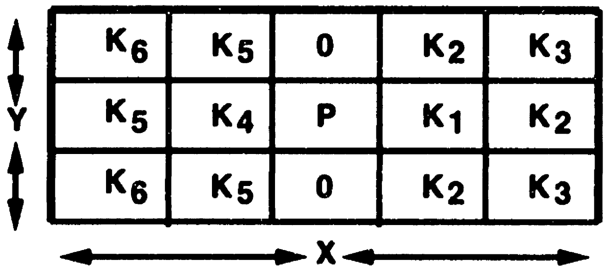


Figure 2.2: Edge Detection Filter Example

The result of this operation is that most edge map values now have a value close to zero except those near sharp edges, which have higher values at locations closest to the edge, as shown in Figure 2.3. The exact values in the edge map will depend on the actual image and the weights used in the edge detection algorithm. The constants K_1 to K_6 are dependent upon the image light levels and algorithm requirements. They are chosen to the maximize the contrast in the edge map between the values at edges and the values which are not at edges. This is done by selecting weights which place the denominator of Equation 2.10 near the value for which the division function's derivative is highest.

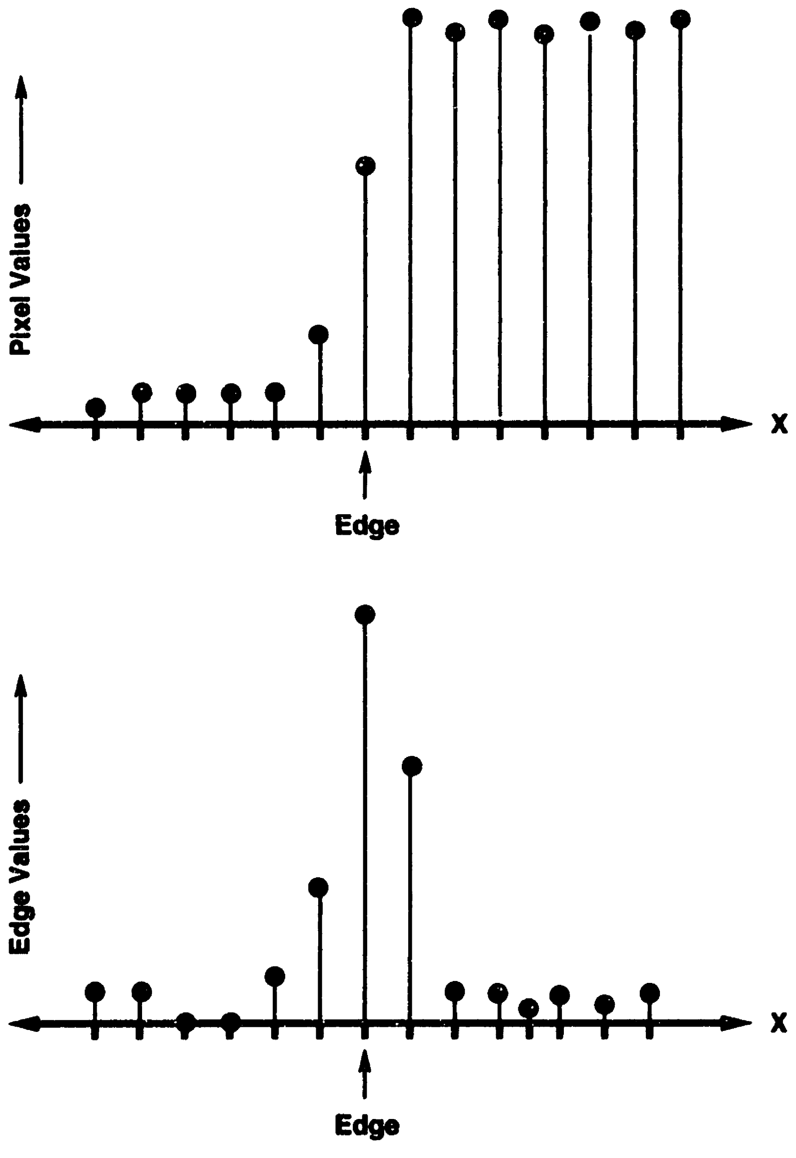


Figure 2.3: Pixels to Edges

2.3 Finding the Disparity

Once the images are converted into edge maps, edges in the right image can be matched with edges in the left image. This is done for each row (i.e., where the y-coordinate is constant) independently. Each row for the left edge map is shifted and then compared to the corresponding row for the right edge map. The number of pixels by which the left row is shifted is the disparity; the comparison is repeated until all possible disparity values are tried. The disparity value for which edge peak values in both edge maps match is the correct disparity. It is of course possible for different edges in the same row to have different disparities since they may correspond to different objects at different distances.

2.4 Trinocular Stereo

The actual algorithm for the ICC system will use trinocular stereo instead of binocular stereo. Trinocular stereo uses a third camera at the midpoint of the baseline. This third camera helps in resolving the correspondence problem. The correspondence problem refers to certain situations in which there is more than one solution to the binocular stereo algorithm. In large images with many edges, this situation almost always exists for some of the edges. An example of this situation is shown in Figure 2.4. The third image is needed to determine whether the objects in the left and right images are from P1 and P2 or P3 and P4. The center image eliminates the P3,P4 solution [44].

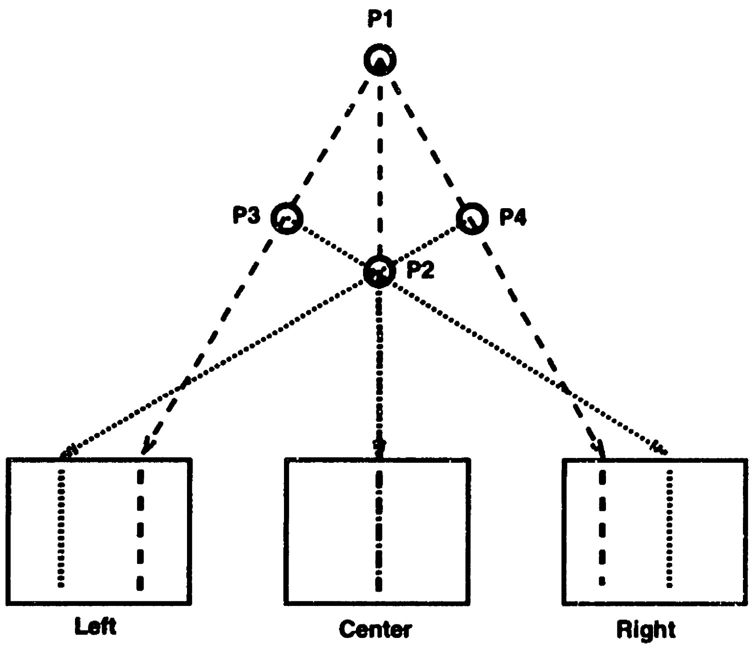


Figure 2.4: Example of Correspondence Problem

2.5 Sub-Pixel Resolution Algorithm

One problem with any stereo algorithm is that the disparity resolution is limited since the algorithm only finds the disparity to the nearest pixel. For an image size of NxN pixels, the range of disparity values is at best 0 to N. This means that there are only N distinct distances that the disparity algorithm can detect. In order to increase the resolution of the algorithm, more pixels must be placed on the imaging chip, which can be expensive.

One way to increase the algorithm's resolution without increasing the number of pixels is to use a sub-pixel resolution algorithm [45]. This algorithm uses the edge values to determine the disparity to a fraction of a pixel. There are a number of ways in which this can be done; the most straightforward way is as follows: At an edge location in the right edge map, the edge peak P_2 (which corresponds to a peak in the left edge map) and the two edge values on either side (P_1 and P_3) are assumed to be three points on a quadratic curve, as shown in Figure 2.5. X_{Actual}, which is the x-coordinate of the curve's peak, can be found from the three peak values by fitting P_1, P_2, and P_3 to a quadratic formula and then finding the x-coordinate of the maximum. This results in the formula:

$$X_{Actual} = \frac{P_2 - .75P_1 - .25P_3}{P_2 - .5P_1 - .5P_3} \tag{2.11}$$

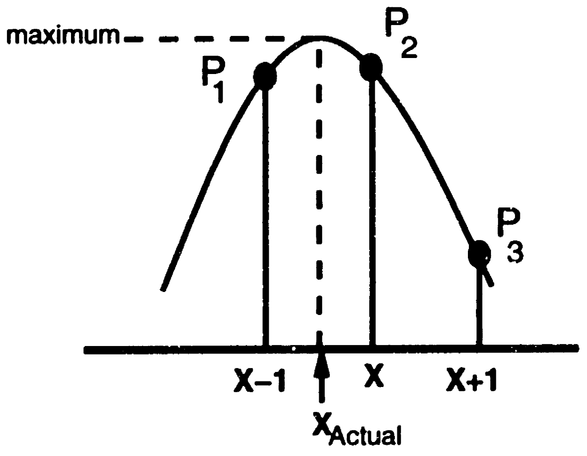


Figure 2.5: Sub-Pixel Resolution Method

Once X_{Actual} is found, the edge location is known to a fraction of a pixel. The same calculation is run on the left edge map. The sub-pixel disparity is then calculated as the difference of the sub-pixel values. Simulations indicate that this method increases the resolution by a factor of at least 4.

There are other algorithms that can be used to perform the sub-pixel resolution calculation [46]. They all involve addition, subtraction, and division operations. In addition, some of them require a logarithm operation.

Chapter 3

ADAP Architecture

This chapter discusses how ADAP represents data, describes the ADAP processor and processor array, explains how an ADAP array can be used, and describes different ways to use an ADAP processor in a vision system.

3.1 Data Representation

The way in which a processor represents numerical information is crucial to its design. Digital processors usually represent information with fixed length bit-strings arranged so that the digital bits are the binary representation of the information. An analog processor such as ADAP can represent information using analog voltages or analog currents.

3.1.1 Analog Current

If the information in an analog system is represented with a current, certain functions are very easy to implement. Addition and subtraction can be implemented using simple current mirrors. Multiplication and division can be implemented using various forms of Gilbert multipliers [47]. Under certain conditions, they can operate at high speeds. However, there are several disadvantages to such a system:

- The current must be constantly reflected through current mirrors to move it around the circuit. To get an accuracy on the order of 1% without expensive laser trimming, they must be very large and slow because of the transistors and cascode structure needed for good matching [48]. Also, their accuracy is dependent upon having well matched transistors, which is not a process characteristic pursued in the most advanced digital processes. For some types of processors, current copiers might be a solution, but they require extra clocking overhead and cannot distribute current to multiple locations.

- Another consideration is that good Gilbert multipliers require bipolar transistors while accurate current mirrors require CMOS transistors. This means that a BiCMOS process, which is more expensive than a CMOS process, is required.

- Finally, in order to be able to distribute information throughout the chip, the current information must be transmitted as the gate voltage of a CMOS transistor in a current mirror. This gate voltage would be routed using CMOS switches. When these switches turn off, charge will be injected into a storage capacitor (either the gate capacitance of a CMOS transistor or a separate capacitor). This charge injection ΔQ divided by the storage capacitance C will produce a voltage error $\Delta V = \Delta Q/C$. This will produce an error of order of $\Delta V/V_{Range}$, where V_{Range} is the range of the transistor's gate voltage. To minimize the impact of this error, a large V_{Range} should be used. A system which represents information using voltage can use a large percentage of the power supply for V_{Range}. ADAP uses 2V (40%) of its power supply range. A current representation can only use the gate voltage range of the current mirror's CMOS transistor. The transistor might be sized to have a gate voltage range of 2V, but this would place the highest gate voltage at about 3V (2V plus about 1V of threshold voltage). Since these current mirrors would need cascode type structures, it would be difficult to place all 4 transistors (2 NMOS and 2 PMOS) with 2V of gate voltage in a 5V power supply.

3.1.2 Analog Voltage

If the information in an analog system is represented as an analog voltage, many of the problems associated with current computation are eliminated. The voltage can be distributed directly throughout the system without any conversion. It can be sampled, held, and processed by circuits whose accuracy can be improved not so much by having well matched devices but by having access to small transistors and many layers of metal.[1] These are characteristics which are often pursued in the most advanced CMOS processes. A larger part of the power supply's voltage range can be used for representing information. Finally, the circuits do not need any bipolar devices; they can be fabricated entirely in a CMOS process. For these reasons, an analog voltage representation method was chosen for ADAP.

3.2 Processor Architecture

3.2.1 ADAP Processor Design

The architecture of an ADAP processor is shown in Figure 3.1. The processor consists of a control register, an analog storage unit, an ALU, and a switch fabric.

The ALU has two analog inputs and one analog output. It can perform addition, subtraction, division, or multiplication. Its functionality is set by the control register. The analog storage unit can store one analog value. The switch fabric routes analog data among the analog storage unit, the ALU, and the 4 I/O lines. The connectivity is determined by the control register. The control unit is programmed by a digital bit-stream that enters at D_IN. This program is loaded once when the chip is powered up, the cell then executes the

[1]This depends on how the system is designed; it is true of the ADAP processor.

same instruction over and over again. The program can be changed, but not continuously on every instruction cycle. The processor design is described in more detail in Chapter 4.

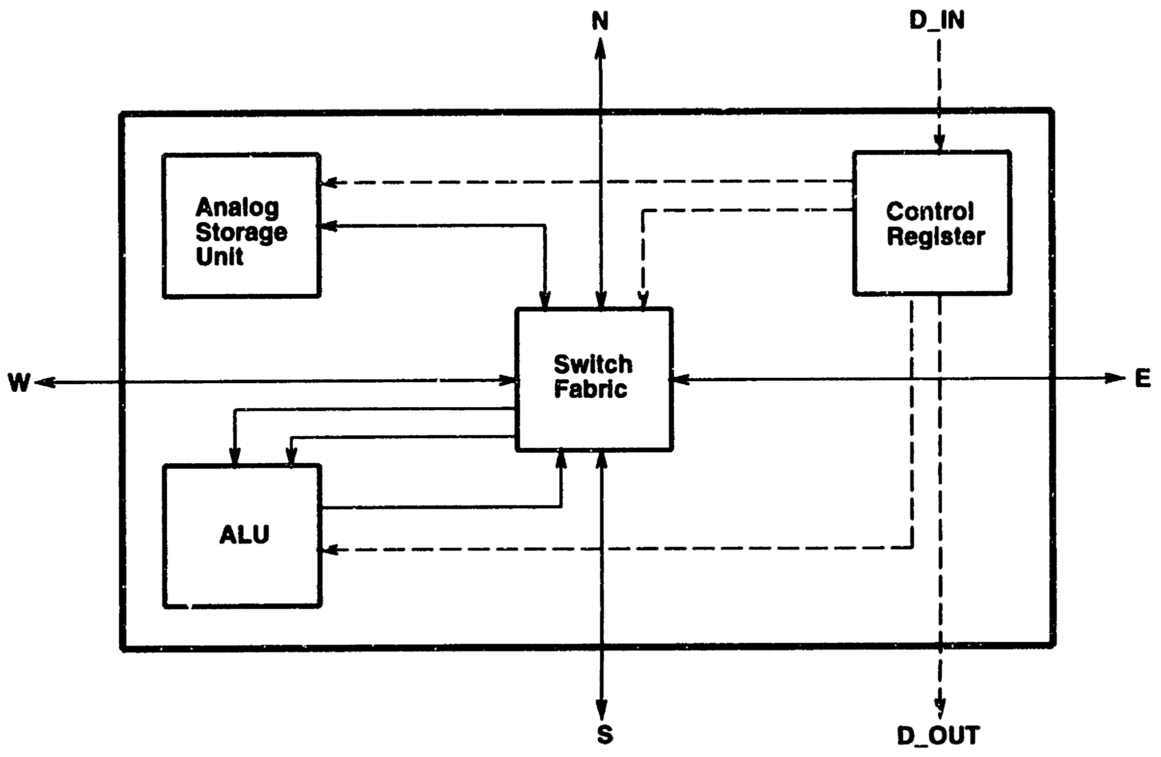


Figure 3.1: ADAP Processor Architecture

3.2.2 Processor Power and Area Efficiencies

The processor is a discrete-time circuit. The storage unit and the ALU sample their inputs at the beginning of every instruction cycle and generate outputs at the the end of the instruction cycle. The instruction cycle actually consists of several clock cycles. This is described in more detail in Section 4.4.1.

Different operations require different sorts of speed, area, and power tradeoffs. There are four main categories of operations to consider: Bit-oriented operations such as bit shifting, AND'ing, and OR'ing; addition and subtraction operations; multiplication; and division.

- A digital processor can perform bit oriented operations. Since ADAP manipulates data in analog form, it cannot perform these operations.

- The ALU in a digital processor can usually perform addition and subtraction as well as bit shifting operations. It can perform these operations in one cycle or several cycles: an N bit-oriented processor will require M cycles to perform an MN-bit addition. For example, a byte-oriented processor will require 2 cycles to perform a 16 bit addition.

Generally, if the bit size is small (smaller N), the area is smaller and the clock speed higher. As N gets larger, the clock speed can be lowered while retaining the same processing speed (i.e. as N goes up, M goes down, so the processing speed MN stays the same), but the area goes up. The power is approximately linearly related to the product of the area and the speed.

An analog processor such as ADAP performs addition in one cycle. Increasing the accuracy of the circuit usually involves larger devices and capacitors to improve matching and reduce charge injection errors; thus, the area of the circuit usually goes up with greater accuracy. The speed is usually dependent upon the available current, so the power goes up with higher speeds.

- For a digital processor, N bit multiplication usually requires N additional cycles or a dedicated multiplier circuit. A dedicated multiplier circuit can require up to N times the area of an addition circuit. An analog processor such as ADAP requires extra cycles to perform multiplication, the accuracy of the operation goes up with the number of clock cycles. Extra accuracy therefore requires more clock cycles per operation. The multiplier can use the adder circuit with some extra area. As with the adder, the accuracy can also be increased by making devices larger, and the processing speed can be increase by using more power. An analog current multiplier requires only one clock cycle but a different circuit.

- For a digital processor, N bit division usually requires either N times more area or N times more cycles to perform than multiplication. Good circuit and algorithm design can usually reduce this to a factor of $\log_2 N$ rather than N. Most analog processors, such as ADAP or a current-oriented circuit, can perform division using the multiplier circuit with a fixed amount of extra area, but no extra power or time is needed.

Efficiency Metric	Addition/Subtraction	Multiplication	Division
Digital (Speed/Power)	1	$1/N$	$1/N \log_2 N$
Digital (Speed/Area)	1	$1/N$	$1/N \log_2 N$
Analog (Speed/Power)	1	$1/N$	$1/N$
Analog (Speed/Area)	1	$1/N$	$1/N$

Table 3.1: Processing Efficiency Scaling for Digital and Analog, N is number of bits

One advantage that ADAP has over a digital processor is that the circuitry used for multiplication at N bits of accuracy can be modified to perform division without N times more area or N times more clock cycles. A digital processor's cycle count, area, or both go up more rapidly when a division function is added. This is summarized in Table 3.1. In this table, the addition/subtraction function is considered the basic operation, and the way in which the area and power efficiencies of the processors scale with bit size is shown [49].

3.3 Array Architecture

3.3.1 ADAP Array Design

The ADAP chip level architecture is shown in Figure 3.2. The figure shows the structure of a 5x5 array although any size that will fit into the available chip area is possible. It is a rectangular array of ADAP processors; each processor can communicate with its four nearest neighbors via analog data lines. (These are the solid lines connecting each cell in the figure.)

The dotted line that begins with D_{in} and snakes throughout the chip is the programming bit-stream. This is how the array program is sent to each cell. This is described in more detail in Section 4.5.

A parallel architecture was chosen for ADAP because of the parallel nature of many vision algorithms. The calculations in these algorithms often involve several pixels in one equation. Therefore, the processor performance is enhanced if it can hold several pixel values and perform operations on all of them in one cycle.

Each cell is connected to its four nearest neighbors by an analog data line. These data lines can be connected to each other within the cell in any possible combination, and to the inputs and outputs of the cell's circuitry..

The $N0 - N4$, $S0 - S4$, $W0 - W4$, and $E0 - E4$ signals are analog I/O lines that go off-chip. They can be either inputs or outputs, depending on how the cells are programmed.

The $O0 - O4$ signals are digital outputs. Since ADAP sends its output to a digital system, ADAP provides the option of performing the A/D function on chip rather than requiring an extra A/D circuit between ADAP and the digital system. It does this by using the output of the A/D in the cell ALU. This is described in more detail in Section 4.4.2.

In addition to the data and control signals, various bias voltages, clock lines, and power supplies are distributed to each cell in the array.

3.3.2 ADAP Array Operation

Although it may be possible in certain cases, it is usually not practical to store an entire image in the ADAP array at once. Instead, the imager will send the pixel values to the ADAP chip in a stream. The ADAP chip will hold a small part (a pixel and its nearest neighbors) of the image at any one time. This will usually be between 9 and 25 pixel values. As a result, ADAP can easily perform operations that require the value of a pixel and nearby pixels, but not operations that require pixel values from widely spaced parts of the image. Essentially, ADAP takes a series of analog inputs, performs a series of arithmetic operations on them, and produces a digital result.

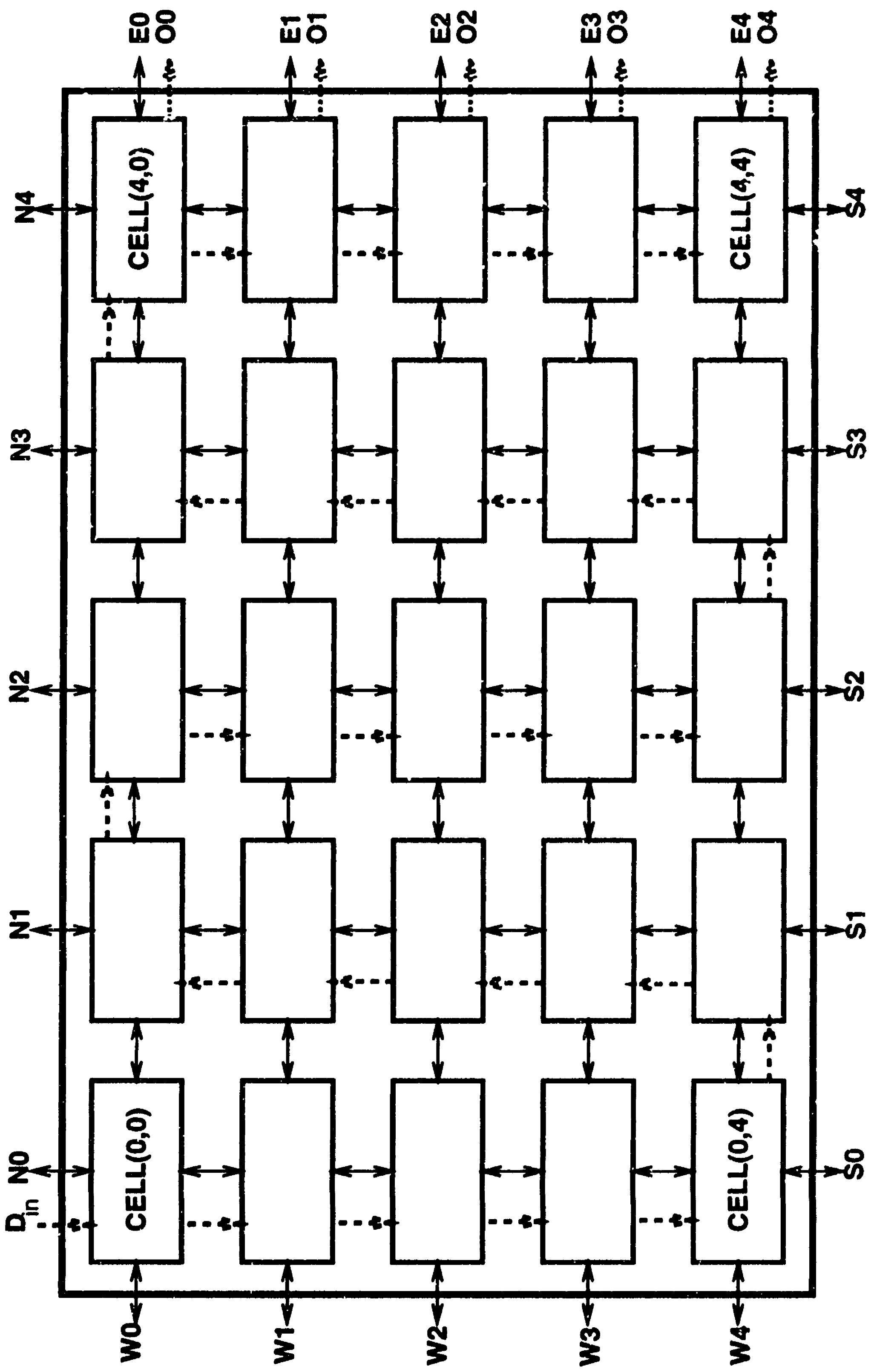


Figure 3.2: ADAP Array Structure

3.3.3 Array Efficiencies

ADAP is a MIMD parallel processor. This means that every processor in ADAP can perform a different function, as opposed to a SIMD processor, in which each processor performs the same function. Since it is possible to use all of the processors at once in a MIMD chip, the silicon area is used very efficiently. Since each processor must perform the same operation in a SIMD processor, some processors will not be used on every instruction cycle. Thus, its area efficiency is lower. For many vision processing algorithms, the parallel nature of the algorithms mean that most processors will be used most of the time. This means that the MIMD/SIMD architectures have similar efficiencies for vision algorithms with high degrees of parallelism. For algorithms without the high degree of parallelism, a MIMD processor has a higher processor utilization, which, in effect, makes its MIPS/area ratio even higher in comparison to a SIMD processor.

Another algorithmic characteristic that should be used in evaluating the efficiency of a processor family is the repeatability of the algorithm. The process of loading instructions into a MIMD processor and some SIMD processors is expensive in terms of area, processing speed, or both since it must be done using either a special instruction bus, which consumes area, or by using clock cycles which could have been used for processing. If the algorithm performs the same group of instructions repeatedly, then these instructions need to be loaded only once, and the time needed for loading the instructions does not hurt the performance. But if the instruction stream is constantly changing, and especially if the instructions are data-dependent, then the MIMD processor will spend a lot of time loading instructions, causing its performance to suffer.

This is summarized in Table 3.2. This table shows the best processor design to use for different classes of algorithms.

Repeatability	Parallelism	
	Parallel	Not Parallel
Repetitive	MIMD,SIMD	MIMD
Not Repetitive	SIMD	SISD

Table 3.2: Best Processor Design for Different Classes of Algorithms

3.4 ADAP System Configurations

There are a number of ways in which ADAP can be used in a vision system. If there is only one imager (camera) in the system, then Figure 3.3 shows the most obvious configuration. In this case, ADAP would perform various image preprocessing algorithms such as edge detection or smoothing.

If there is more than one imager in the system, then ADAP can be used in several different configurations. Some of these are shown in Figure 3.4. For the first configuration, ADAP performs an image preprocessing function for each imager (such as edge detection), and the

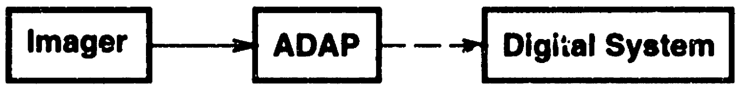


Figure 3.3: ADAP System Configuration: One Imager

digital system performs operations that require data from both images (such as disparity calculation). In the second configuration, ADAP not only performs image preprocessing functions, but also performs functions using data from both images. The last configuration is one where some of ADAP's functions require data from both images.

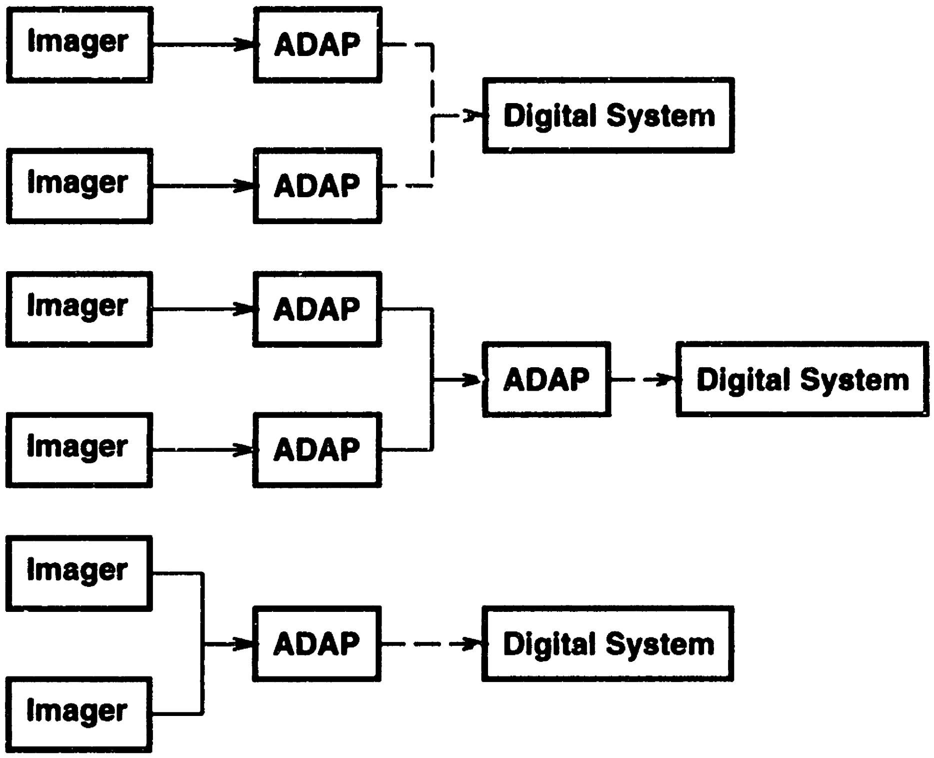


Figure 3.4: ADAP System Configurations: Two Imagers

A possible configuration for a stereo system is shown in Figure 3.5. In this system, the first set of ADAPs is used for edge detection, and the second set of ADAP's is used for calculating the sub-pixel resolution values. The digital system performs the remaining part of the stereo algorithm (disparity determination and distance from disparity calculation).

In all of the configurations described above, if the necessary ADAP array cannot fit onto one chip, several ADAP chips can be connected together. However, for the maximum processing speed, it is best to perform all of the ADAP functions on one chip.

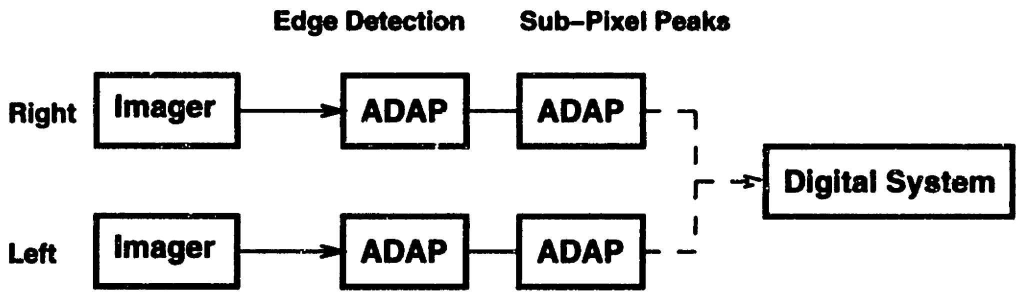


Figure 3.5: ADAP System Configuration: Stereo Vision System

Chapter 4

Circuit Design

This chapter describes ADAP processor cell and its constituent circuits in detail. ADAP was fabricated in HP's 0.8μm triple metal CMOS process. ADAP uses a single ended 5V power supply for both the digital and analog circuits.

4.1 Cell Processor Design

A detailed block diagram of an ADAP processor is shown in Figure 4.1. The functionality of the processor was described in Section 3.2. The control registers and their control logic are described in Section 4.5. The control registers output bits control the operation of the ALU, which consists of input and output sample and holds, an A/D, a D/A, a shift register, and control logic. It is described in detail in Section 4.4. The analog storage unit is a simple analog sample and hold, which is described in Section 4.3. The data from the cell's nearest neighbors, the ALU's inputs and outputs, and the sample and hold's inputs and outputs are all routed through the switch fabric. In addition, each row has an extra data line, called X, which is routed through the switch fabric. This allows a constant to be distributed to every cell in a row without using up the cells' regular routing. This feature was added because many vision algorithms involve sums of pixel values multiplied by constants. The ability to distribute the constant without using extra area increases the overall area efficiency of the processor. Also, the output of the ALU, AO, is also routed directly to the southern cell (where the incoming signal is named AI). The switch fabric is described in Section 4.2. The cell timing is described briefly in Section 4.4.1 and in detail in Appendix B.

4.2 Switch Fabric

Table 4.1 shows all of the possible connections in the switch fabric. An X in a box shows that the two signals can be connected, a 0 means that the connection cannot be made. A1 and A2 are the ALU inputs, AO is the ALU output. MI and MO are the storage unit's input and output, as shown in Figure 4.1. There are 37 possible connections, but some of them,

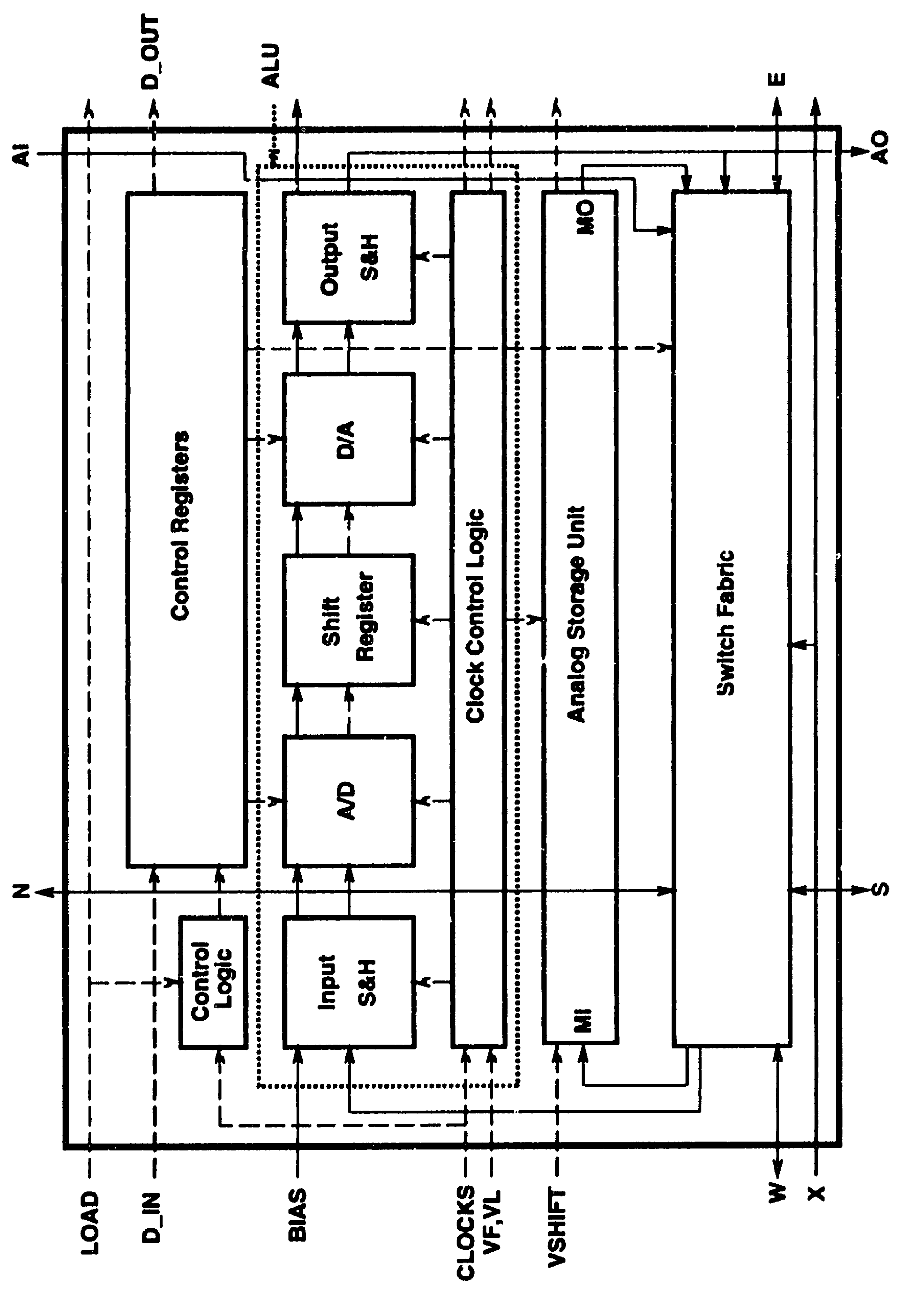


Figure 4.1: ADAP Cell Block Diagram

such as N-S and S-N, are redundant, resulting in only 31 independent connections.

The switch fabric consists of 31 NMOS pass transistors. The gate of each transistor is connected to an output of the control register. The source and drain of the transistor are connected to the two signals which can be connected. Thus, a high output bit from the control register will cause the NMOS transistor to connect the source and drain of the transistor, thus connecting the two signals. A low output bit will keep the two signals disconnected.

Although it is possible to have every connection made by setting every bit in the control register to 1, this would probably not make sense because this would connect several outputs together. Care must be taken in programming the cell to make only the desired connections.

Signal Name	A0	MO	N	S	E	W	X	AI
A1	0	X	X	X	X	X	X	X
A2	0	X	X	X	X	X	0	0
MI	X	0	X	X	X	X	0	0
N	X	X		X	X	X	0	0
S	X	X	X		X	X	0	0
E	X	X	X	X		X	0	0
W	X	X	X	X	X		0	0

Table 4.1: Switch Fabric Possible Connections

Each transistor in the switch fabric is 1.6μm wide and 0.8μm long, which is close to the minimum size, in order to minimize the area. Since the transistor adds a resistance in series with the capacitive load being driven, it creates an RC delay which places a limitation on how fast a processor can communicate with its neighbors. If there are too many pass gates in series between the signal source and its destination, the signal will not be able to settle to its final value by the end of the clock cycle. Excessive parasitic capacitance and resistance from the routing can have a similar effect on performance. This can be overcome by making the pass transistors wider (unless it is the parasitic capacitance of the transistors which dominates, in which case increasing the transistor's size does not reduce the delay.), by increasing the digital power supply voltage, or by increasing the current drive and speed of the opamp driving the output signal. Simulations indicated that the cell output drivers would be able to drive 3 inputs one cell away to within one LSB at a processor speed of 2 MIPS.

4.3 Analog Storage Unit

The analog storage unit is a sample and hold circuit (SH); its operation is shown in Figure 4.2. The capacitor for this circuit is 124fF. In the sample phase, the charge stored on the capacitor is:

$$Q = (V_{IN} - V_-)C \tag{4.1}$$

In the hold phase, the top plate of the capacitor is then attached to the output of the opamp. The voltage across the capacitor is still:

$$V_C = V_{IN} - V_- \tag{4.2}$$

This voltage is then added to the voltage at the negative input, V_-, giving:

$$V_{OUT} = V_{IN} \tag{4.3}$$

Thus, any opamp offset cancels out exactly, and there is no capacitor matching needed. The only error comes from charge injection and the finite gain of the opamp. This is discussed in detail in Appendix A.

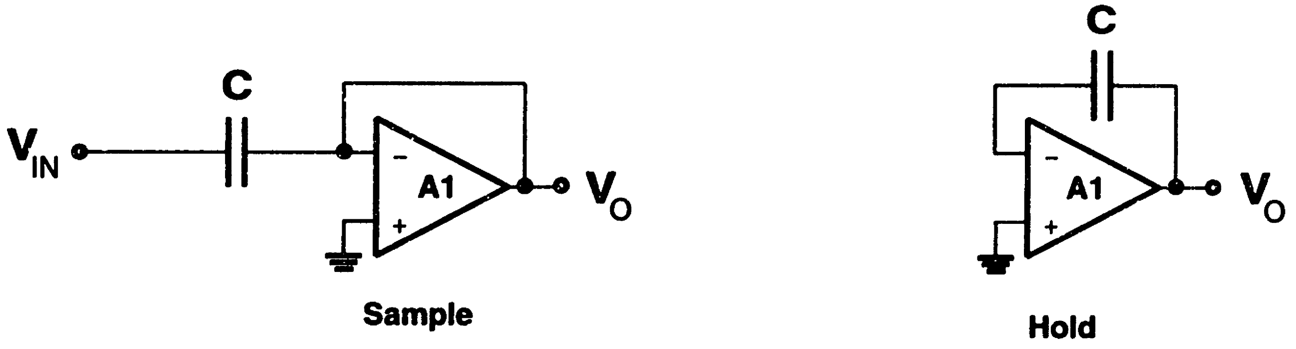


Figure 4.2: Sample and Hold Operation

One potential problem with the SH circuit is that when the circuit enters the hold phase, the output cannot immediately swing to its final value. Instead, it starts out at its sample mode value of V_+. It then slews towards its final value. This means that if the voltage across the capacitor is its maximum possible value ($V_{IN,Maximum} - V_-$), the voltage at the negative input node (V_-) can be pushed down to $V_+ - (V_{IN,Maximum} - V_-) = V_+ - (V_{IN,Maximum} - V_+) = 2V_+ - V_{IN,Maximum} = -V_{IN,Maximum}$[1]. If this value falls far enough below the substrate voltage, charge can leak into the node from the p-type substrate, which is at 0V. In addition, if the V_- node falls more than a threshold voltage below 0V, the feedback switch around the opamp, whose gate is at 0V, can turn back on and leak charge into V_- as well. This problem did occur for ADAP; it was solved by connecting the bottom plate of the capacitor to V_- instead of the top plate. The bottom plate of the capacitor has a parasitic capacitance C_P of about 82fF to the substrate, as shown in Figure 4.3. The parasitic capacitance is large enough to absorb enough of the charge stored at V_- to prevent the voltage at V_- from falling far enough to allow charge leakage from the substrate or the feedback switch.

One side affect of this strategy is to reduce the bandwidth of the circuit because of the feedback factor, which causes the circuit to settle more slowly than if the capacitor was reversed. Also, the circuit is more susceptible to noise coupling from the substrate.

The analog storage unit's input and output are connected to the switch fabric, through which they can be connected to any other signal connected to the switch fabric (the ALU

[1]Note that in Figure 4.2, V_+ is 0; however, in the actual implementation, V_+ will be held at approximately 0.945V. This is explained further in Section 4.4.7.

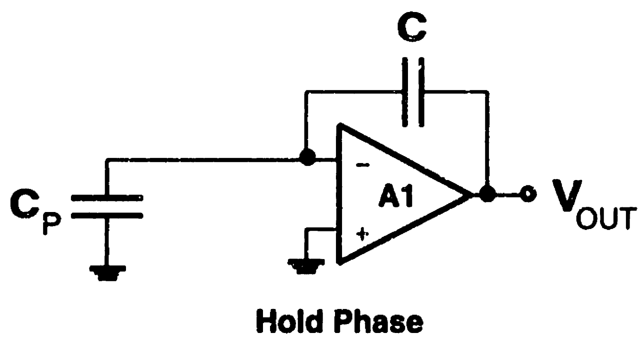


Figure 4.3: Sample and Hold Parasitic Capacitance Method

inputs and outputs, and the signals from the four neighboring processors). All of the storage units sample their inputs during the first phase of an instruction cycle. Some of these inputs will be driven by the output of the storage unit on another processor. As a result, the storage unit needs the ability to sample its inputs *and* drive its output at the same time. To do this, it uses two SH circuits in series. While the first one is sampling a new input, the second one is driving its output. Then, during the clock cycles when the analog storage unit does not need to drive its output, the second SH circuit samples the output of the first SH circuit. This is described in detail in Appendix B.

When the second SH circuit is sampling, its output is disconnected (by turning off a pass gate) from the processor cell's outputs. The reason for this is that the large capacitive load associated with the output reduces the slew rate of the opamp, making it difficult for the opamp's output to slew down to V_+ in one clock cycle. Disconnecting the output load eliminates this problem. When driving the load, the SH circuit has two clock cycles for its output to settle.

4.4 ALU

The ADAP ALU uses a new circuit for performing arithmetic. This circuit uses an A/D followed by a D/A to perform a calculation, as shown in Figure 4.4. Assume that each sub-circuit has N bits of resolution. The output of the A/D, D_{MID}, equals $\frac{V_{IN}}{V_{REF1}}2^N$. The output of the D/A, V_{OUT}, is $\frac{D_{MID}}{2^N}V_{REF2}$. When the two equations are combined, the 2^N factor drops out, and the result is:

$$V_{OUT} = \frac{V_{IN}V_{REF2}}{V_{REF1}} \tag{4.4}$$

Thus, the output is a function of a division and a multiplication. By modifying the input of the A/D, the circuit can also perform an addition and a subtraction. Thus, with one circuit, four arithmetic operations can be performed. Section 4.4.9 will describe in detail how this is done.

Although Equation 4.4 is continuous, the circuit actually quantizes the information. This is discussed in Appendix A.2.

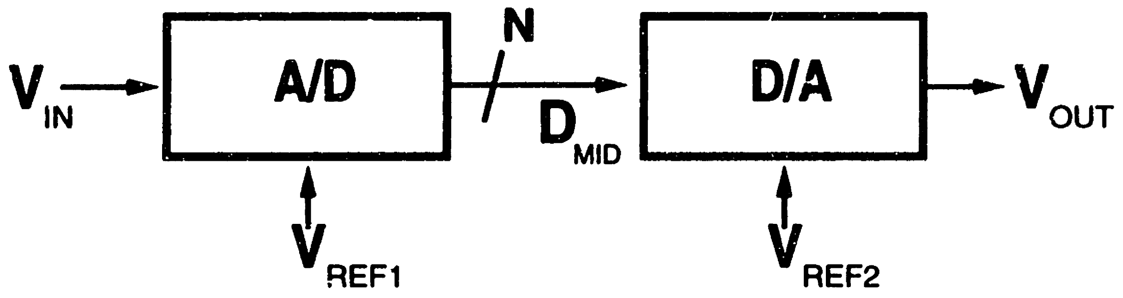


Figure 4.4: A/D - D/A Processor Concept

4.4.1 ALU Timing

ADAP is clocked by the two phase non-overlapping clocks ϕ_1 and ϕ_2. These are produced by a system clock, as shown in Figure 4.5. Each cycle of the system clock has two phases, one where ϕ_1 is high, and one where ϕ_2 is high. The ADAP circuits perform different actions on different phases. Thus, in one cycle of the system clock, the ADAP circuits will have performed two actions, one while ϕ_1 was high, and one while ϕ_2 was high. Since each ADAP instruction cycle consists of 10 different actions, it uses 10 phases, or 5 system clock cycles. Since ADAP's instruction cycle uses 5 cycles of the master clock, its processing speed is one fifth of the system clock speed. For example, if ADAP is clocked at 5MHz, ADAP will process data at 1.0 MIPS. For the rest of this chapter, unless otherwise stated, a cycle will refer to a phase-cycle, i.e., either ϕ_1 or ϕ_2 being high. More timing details are given in Appendix B.

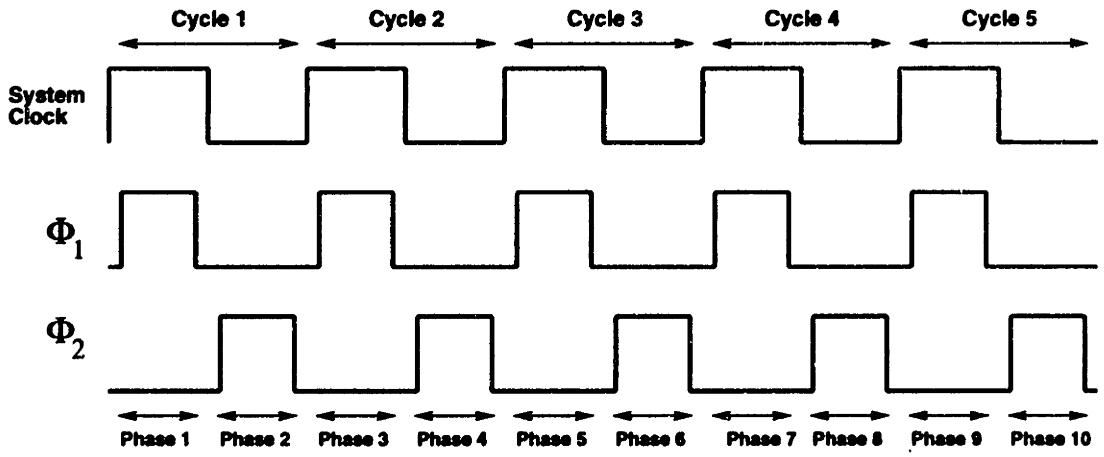


Figure 4.5: One ALU Instruction Cycle = Five Clock Cycles = Ten Phases

4.4.2 Design of the A/D

A cyclic A/D was selected for the ADAP ALU because it used the smallest area for the necessary speed, power, and accuracy requirements.

The A/D algorithm is shown in Figure 4.6 [50]. Its digital output is serial, starting with the MSB. It has a small area requirement since it uses the same circuit for each bit.

The A/D samples its input and compares it against a voltage reference V_{REF}. This voltage is half of the full scale voltage ($V_{REF} = V_{FULL\ SCALE}/2$). If the input is greater than V_{REF}, the MSB is set to one and V_{REF} is subtracted from V_{NOW}; otherwise, the MSB is set to zero and V_{NOW} is not modified. V_{SUM} is then multiplied by 2 to get V_{NEXT}. At the end of the first cycle:

$$V_{NEXT} = 2(V_{IN} - V_{REF}) \tag{4.5}$$

if $V_{IN} > V_{REF}$, or:

$$V_{NEXT} = V_{IN} \tag{4.6}$$

if $V_{IN} < V_{REF}$.

The cycle is repeated with V_{NEXT} replacing V_{IN} on the subsequent cycles:

$$V_{NOW,\ cycle\ N+1} = V_{NEXT,\ cycle\ N} \tag{4.7}$$

On each cycle, the A/D determines the MSB of its input, removes that information from its input, and then re-scales the input. The A/D generates one bit for every cycle. For 8 bits of resolution, the A/D must be clocked for 8 cycles. The other 2 cycles of the ADAP instruction cycle are used for setting up the arithmetic functions, as explained in Section 4.4.9.

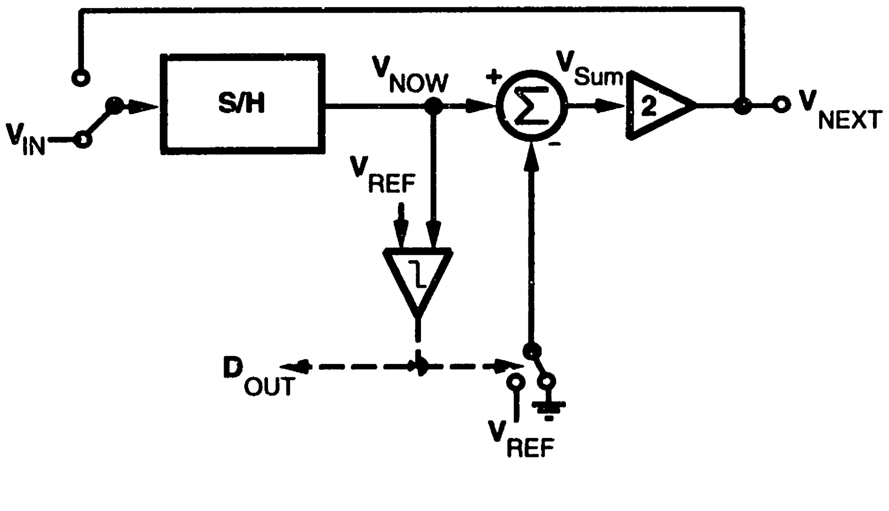


Figure 4.6: A/D Algorithm

The A/D schematic is shown in Figure 4.7; it is implemented with two opamps, four capacitors, NMOS switches, and a comparator. Each capacitor is 53fF; this value was chosen to provide a simulated accuracy of 8 bits. In this circuit schematic, V_G is an offset voltage

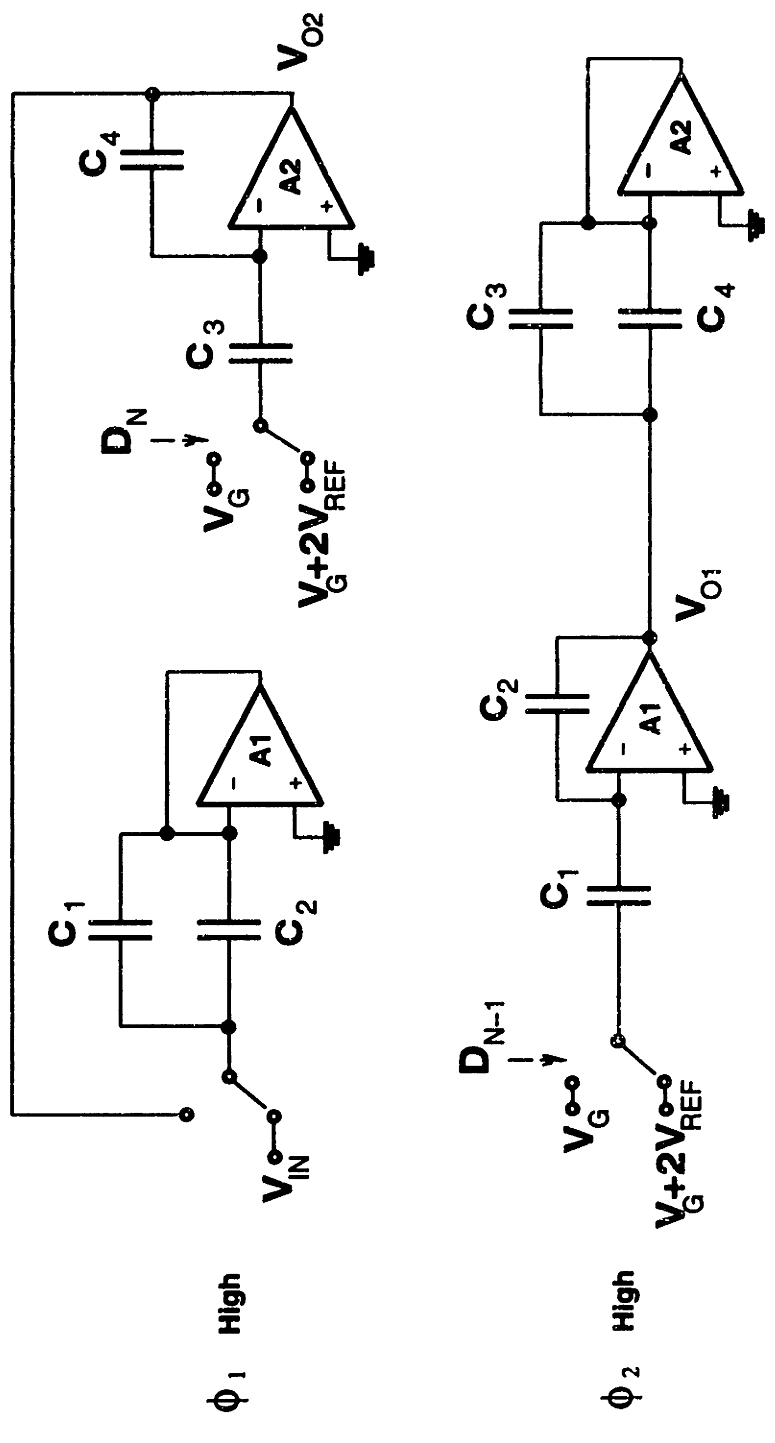


Figure 4.7: A/D Schematic

used to place the analog signal in the correct voltage range. When ϕ_1 is high, C_1 and C_2 sample the output of opamp A2 (or V_{IN} if this is the first cycle) while opamp A2 performs the multiply-by-2 and reference subtraction of Equations 4.5 and 4.6. When ϕ_2 goes high,

C_3 and C_4 sample the output of opamp A1 while opamp A1 performs the multiply-by-2 and reference subtraction function.

For simplicity, the operation of the comparator is not shown in this diagram. The operation of the comparator is described in Section 4.4.3.

The multiplication and subtraction occur as follows. When ϕ_1 is high, charge is collected on C_1 and C_2. This charge is:

$$Q = (V_{O2} - V_+)(C_1 + C_2) = V_{O2}(C_1 + C_2) \tag{4.8}$$

When ϕ_2 goes high, the top plate of C_2 is attached to the output of opamp A1, and the top plate of C_1 is attached to either V_G or to $V_G + 2V_{REF}$ depending upon the output of the comparator. Thus the charge across C_2 is either:

$$Q = V_{O2}(C_1 + C_2) - (V_G - 0)C_1 \tag{4.9}$$

or

$$Q = V_{O2}(C_1 + C_2) - [(V_G + 2V_{REF}) - 0]C_1 \tag{4.10}$$

Simplifying terms, and dividing the charge by C_2 to get the voltage across C_2, yields:

$$V_{C2} = 2V_{O2} - 2V_G + V_G \tag{4.11}$$

or

$$V_{C2} = 2V_{O2} - 2V_G + V_G - 2V_{REF} \tag{4.12}$$

Rearranging terms and adding this to the voltage at the negative input of opamp A1, $V_- = V_+ = 0$, gives:

$$V_{O1} = 2(V_{O2} - V_G) + V_G \tag{4.13}$$

or

$$V_{O1} = 2((V_{O2} - V_G) - V_{REF}) + V_G \tag{4.14}$$

The circuit thus performs the desired function. It takes V_{O2}, subtracts the offset V_G, subtracts either V_{REF} or nothing, multiplies by 2, and then adds the offset V_G back on. The same process is performed by opamp A2 in its multiply/subtract phase. The effects of charge injection, opamp offset, capacitor mismatch, and finite opamp gain are discussed in Appendix A.

4.4.3 Comparator Operation

The comparison phase of the A/D cycle is shown in Figure 4.8. During the time period when both ϕ_1 and ϕ_2 are low, the top plates of C_1 and C_2 are connected to $V_G + V_{REF}$. If the voltage across the capacitors was greater than V_{REF}, then the negative node of the opamp will be pushed below V_+, the output of the opamp will go up towards 5V, and the comparator output will go up. The opposite happens if the voltage across the capacitors is

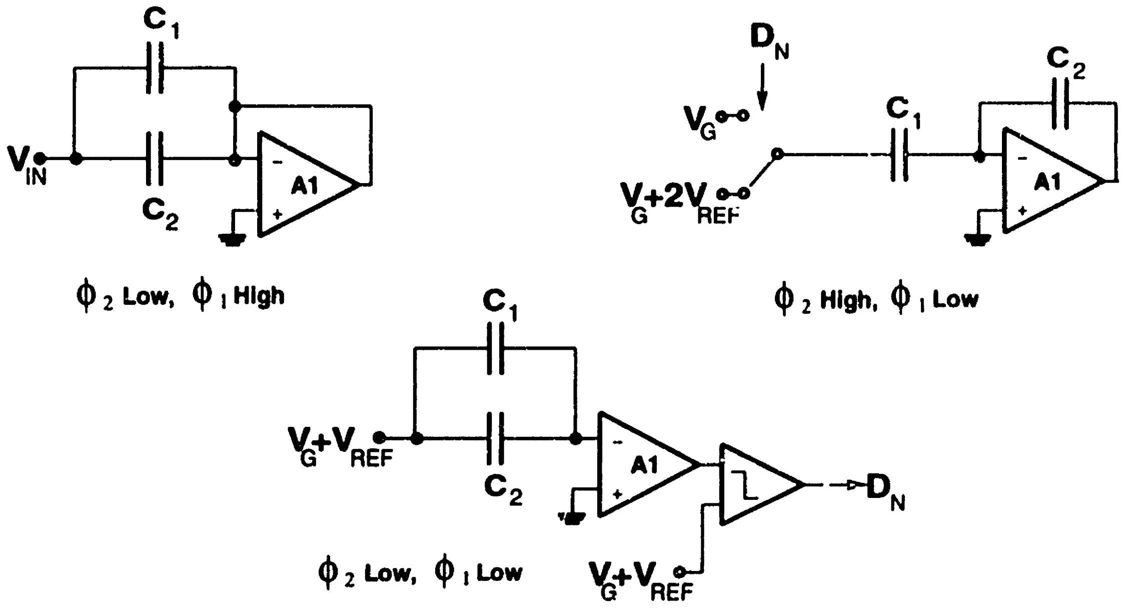


Figure 4.8: Comparator Operation

less than V_{REF}. The comparator output is valid when ϕ_2 goes high. The comparator input is connected to the output of the opamps through two NMOS switches which alternate connecting the output of A1 (at the end of ϕ_1 clock phases) or A2 (at the end of ϕ_2 clock phases) to the comparator input. The opamps are thus used as preamps for the comparator. The comparison is done in this way to simplify the timing of the A/D circuit; it also saves area by eliminating the need for a separate preamp for the comparator.

One potential problem with this circuit is that the output voltage of the opamp might need to start out at V_+ and slew up past $V_G + V_{REF}$ in the very short non-overlapping period. To eliminate this problem, the output of the opamp is connected through an NMOS pass transistor to a bias voltage at $V_G + V_{REF}$ for about 0.5ns. This pulls the output to $V_G + V_{REF}$. The opamp then has the entire non-overlapping period to slew above or below this voltage. The gate of the NMOS pass transistor is the output of a digital circuit which always has a low output but has a momentary glitch to a high value when ϕ_1 or ϕ_2 go low. Note that the timing of this glitch will vary slightly with process variations; the non-overlapping period must be long enough to handle the longest expected glitch time.

4.4.4 Design of the D/A

A cyclic D/A was selected for the ALU because it used the smallest amount of area for the necessary speed, power, and accuracy requirements.

The D/A algorithm is shown in Figure 4.9. Its digital input is serial, starting with the LSB. It has a small area requirement since it uses the same circuit for each bit. The D/A starts with a state voltage (V_{STATE}) of zero. On each cycle, the D/A divides V_{STATE} by two. Before dividing by two, if the digital input bit is one, it adds the reference voltage

$V_{FULL\ SCALE}$ to V_{STATE}:

$$V_{OUT} = V_{STATE}/2 \tag{4.15}$$

if $D_{IN} = 0$, or else:

$$V_{OUT} = (V_{STATE} + V_{FULL\ SCALE})/2 \tag{4.16}$$

if $D_{IN} = 1$.

V_{OUT} becomes V_{STATE} on the next cycle. Thus after N cycles, $V_{FULL\ SCALE}$ has been divided by 2^N for D_N (which is the LSB), 2^{N-1} for the next bit, and so on. The final result is:

$$V_{OUT} = V_{FULL\ SCALE}(D_N/2^N + D_{N-1}/2^{N-1} + \cdots + D_1/2^1) \tag{4.17}$$

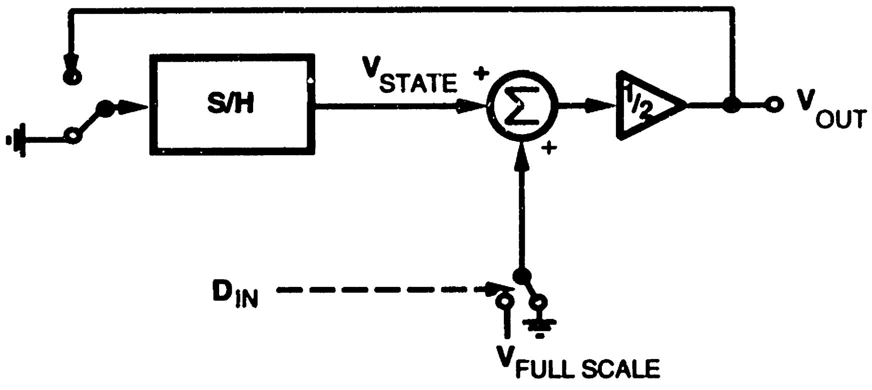


Figure 4.9: D/A Algorithm

For 8 bits of resolution, the D/A must be clocked for 8 cycles The other 2 cycles of the ADAP instruction cycle are used for setting up the arithmetic functions, as explained in Section 4.4.9.

The D/A schematic is shown in Figure 4.10; it is implemented with two opamps, four capacitors, and NMOS switches. Each capacitor is 53fF; this value was chosen to provide a simulated accuracy of 8 bits. In this circuit diagram, V_G is an offset voltage used to place the analog signal in the opamps' high gain output range. When ϕ_1 is high, C_1 samples the output of opamp A2 (or V_G if it is the first cycle) while C_2 samples either V_G or $V_G + 2V_{REF}$. Opamp A2 performs a divide-by- two and sum operation. When ϕ_2 is high, opamp A1 performs a divide-by-two and sum operation while C_3 and C_4 are in sample mode.

The adding and divide by two operations work as follows. When ϕ_1 is high, the charge on C_1 is $(V_{O2} - V_+)C_1 = V_{O2}C_1$. The charge on C_2 is, depending on the bit value, either:

$$Q = (V_G - V_+)C_2 = V_G C_2 \tag{4.18}$$

or

$$Q = [(V_G + 2V_{REF}) - V_+]C_2 = (V_G + 2V_{REF})C_2 \tag{4.19}$$

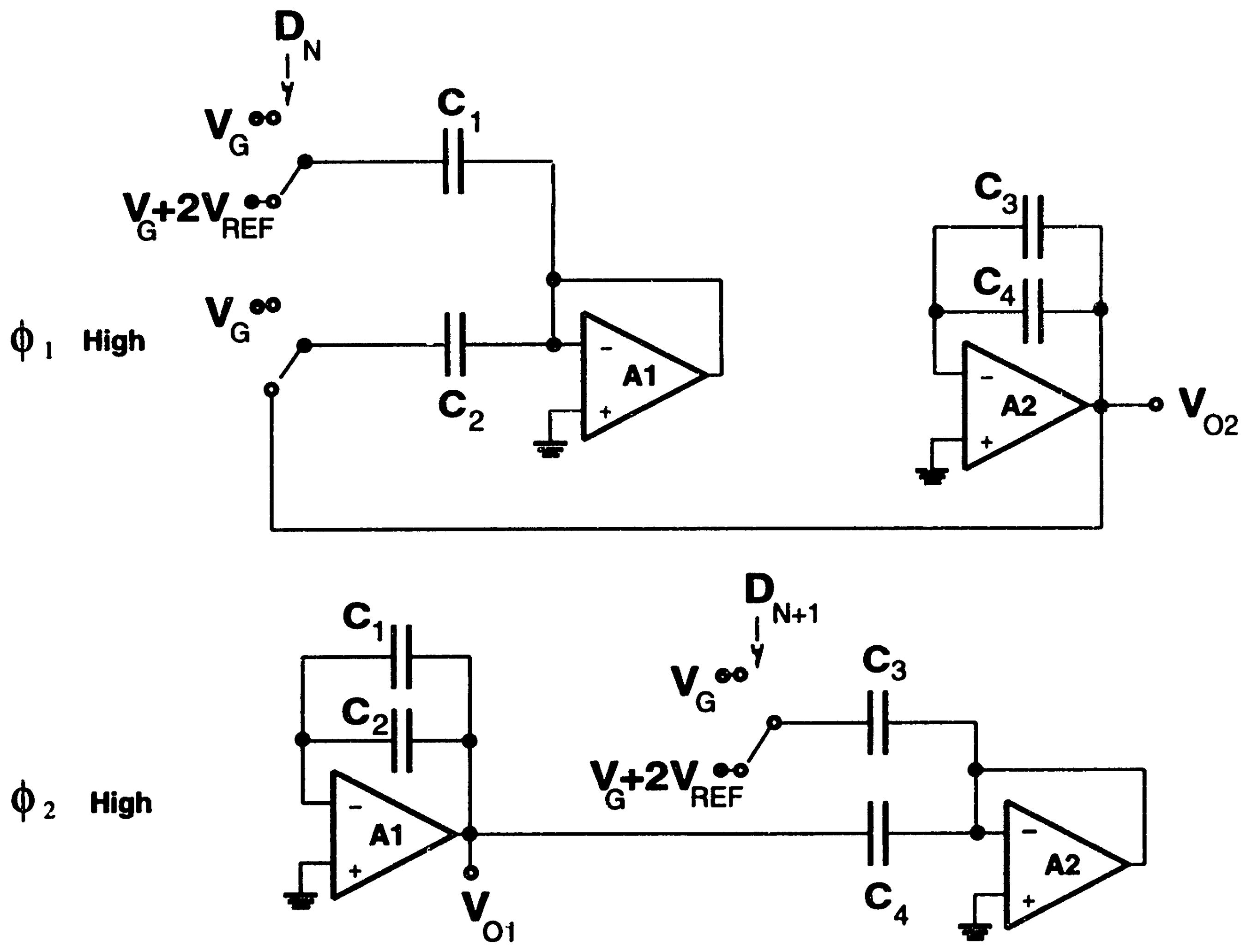


Figure 4.10: D/A Schematic

When ϕ_2 goes high, the top plates of C_1 and C_2 are connected to the output of opamp A1. The total charge on them is now:

$$Q = V_G C_2 + V_{O2} C_1 \tag{4.20}$$

or

$$Q = (V_G + 2V_{REF})C_2 + V_{O2} C_1 \tag{4.21}$$

Simplifying terms and dividing by $(C_1 + C_2)$ to get the voltage across the capacitors gives:

$$V_{C_{1,2}} = (V_{O2} - V_G)/2 + V_G \tag{4.22}$$

or

$$V_{C_{1,2}} = ((V_{O2} - V_G) + 2V_{REF})/2 + V_G \tag{4.23}$$

Adding this to the voltage at the negative input of opamp A1, $V_- = V_+ = 0$, gives:

$$V_{O1} = (V_{O2} - V_G)/2 + V_G \qquad (4.24)$$

or

$$V_{O1} = ((V_{O2} - V_G) + 2V_{REF})/2 + V_G \qquad (4.25)$$

The circuit thus performs the desired function. It takes the state voltage, subtracts the offset voltage V_G, adds either the full scale voltage ($2V_{REF}$) or nothing, divides by 2, and adds the offset V_G back. The effects of charge injection, opamp offset, capacitor mismatch, and finite opamp gain are discussed in Appendix A.

The ADAP processor would be simpler and smaller if the D/A could begin its operation with the MSB rather than the LSB. However, this would require a circuit which constantly multiplied, rather than divided, a voltage by two. This would increase the error by approximately 2^{N-1} because the charge injection error would be multiplied by two every cycle rather than divided by two.

4.4.5 Digital Buffer Design

Because the A/D's output stream is MSB-first while the D/A needs its data LSB-first, there is a bidirectional 8 bit buffer between the two circuits to hold the A/D output and feed it to the D/A. The cell operations are pipelined so that both the A/D and D/A operate during every instruction cycle. This buffer is an 8 bit bidirectional shift register. Since this register must shift data for both the ϕ_1 and ϕ_2 cycles, it uses a single-phase clock register design as presented in [51]. The clock signal used to clock the shift register is $\overline{\phi_{12}}$, which is equal to $\overline{\phi_1 + \phi_2}$.

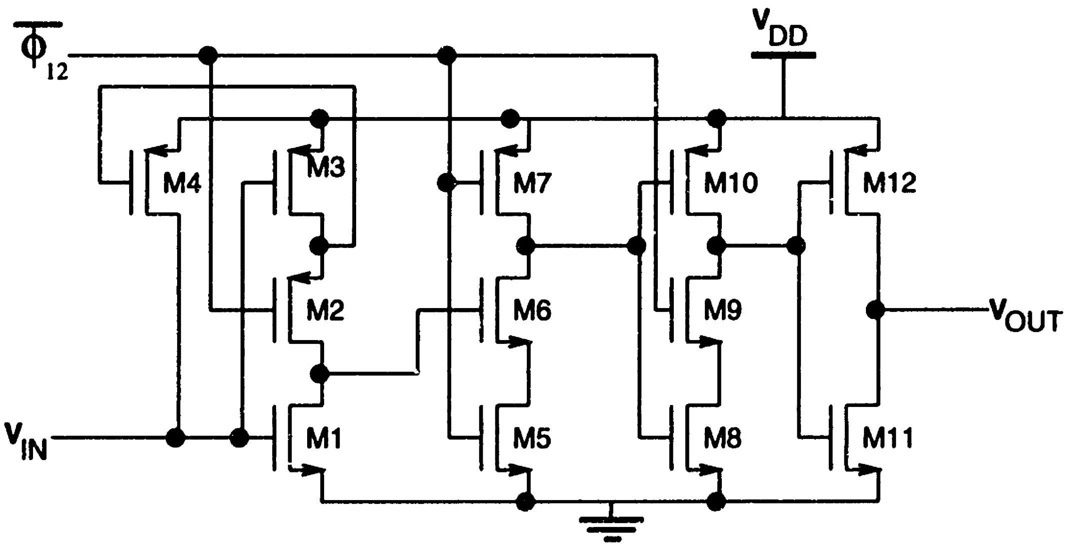


Figure 4.11: Shift Register Schematic

Name	Size (μm/μm)	Name	Size (μm/μm)	Name	Size(μm/μm)
M1	1.2/3.2	M5	1.6/0.8	M9	1.6/0.8
M2	1.6/0.8	M6	1.6/0.8	M10	2/0.8
M3	1.6/0.8	M7	1.2/1.6	M11	1.6/0.8
M4	1.2/2	M8	1.6/0.8	M12	1.6/0.8

Table 4.2: Shift Register Transistor Sizes

The circuit schematic is shown in Figure 4.11. The purpose of M4 is to act as a pull-up device for V_{IN}, which is the output of an NMOS pass gate. The transistor sizes are shown in Table 4.2. The timing of the register is detailed in Appendix B.4.

The shift register is used as shown in Figure 4.12. Depending on the value of DIR, data is fed into either IF or IB and taken out of either OF or OB. A bidirectional shift register is formed by chaining 8 of them together. The DIR signal flips at the beginning of every instruction cycle. The DIR signal is a function of V_F as shown in Figure 4.13.

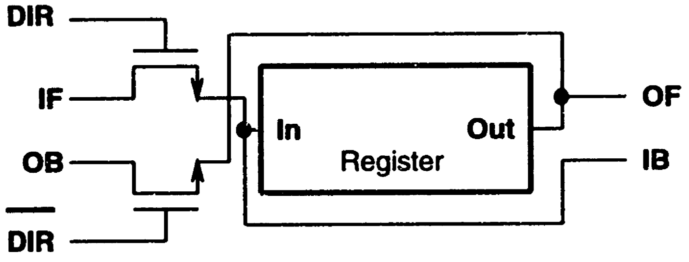


Figure 4.12: Shift Register Configuration

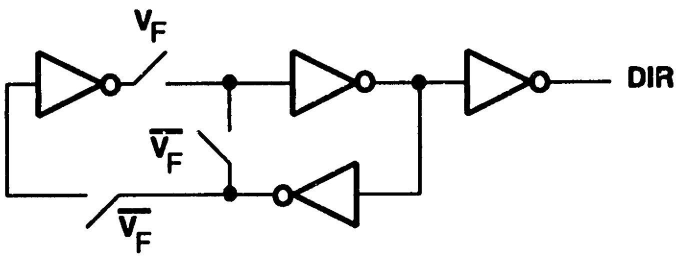


Figure 4.13: DIR Circuit

4.4.6 ALU Sample and Holds

The analog storage unit stores analog data for the processor. The ALU requires additional SH circuits to hold temporary values during its operation.The ALU sample and hold circuits use the same design as the analog storage unit described in Section 4.3 except their storage capacitor is only 93fF since their accuracy requirement is not quite as high. Also, they need only one SH stage since they do not need the ability to sample and hold at the same time. There are three input SH circuits used by the A/D. One holds V_2 for all of the arithmetic operations except addition[2]. The other two hold the A/D's V_{REF} and V_{REF2} during the division operation. There are two output SH circuits for the D/A. One holds V_2 for the multiplication and division operations while the other is used to drive the ALU's output.

4.4.7 Opamp Design

The opamp design is shown in Figure 4.14. The architecture was chosen for its simplicity and low area. It has a 5V power supply and V_{Bias} is 2.47V. Note that in the previous circuit diagrams, V_+ was shown connected to ground for simplicity; however, V_+ is *actually* always connected to 0.945V. This opamp has very large offset and a very small common mode range; however, the circuits in which the opamp operates cancel out the offset, and the common mode never changes. To save area, a second opamp shares M1, M3, M5, and M7 with the first opamp, so that only 12 transistors are needed for two opamps. The transistor sizes are shown in Table 4.3.

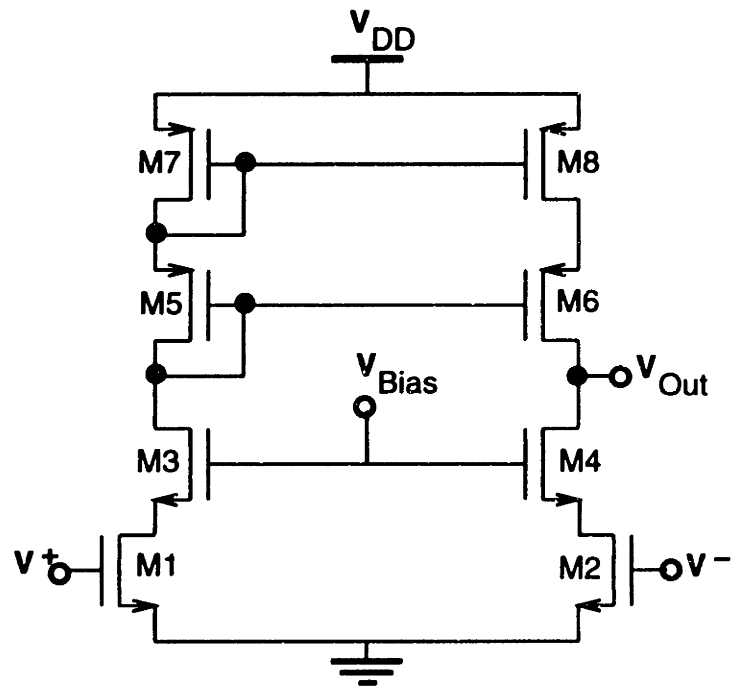


Figure 4.14: Opamp Schematic

The transistor sizes were picked in the following manner. First, the opamp was simulated using minimum size transistors. M1 and M2 were then widened, and the lengths of all transistors increased, to get a gain of about 3,000 and a unity gain frequency of about

[2]V_2 is described in Table 4.6.

Name	Size (μm/μm)	Name	Size (μm/μm)
M1	9.6/1.2	M5	10/1.6
M2	9.6/1.2	M6	10/1.6
M3	2/1.8	M7	10/1.6
M4	2/1.8	M8	10/1.6

Table 4.3: Opamp Transistor Sizes

40MHz. The load capacitance, including parasitics, of about 700fF was included in this calculation. The ratios of of M3, M4, M5, M6, M7, and M8 were then increased to get the necessary output swing. This process was iterated several times to get the simulated opamp specifications shown in Table 4.4.

Name	Value
Output Swing	0.35 to 5V
DC Gain	3,000
Unity Gain Bandwidth	42MHz
Slew Rate	53V/μs
7-Bit Settling Time	45ns
Phase Margin	86°
PSRR+	65dB at 1MHz
PSRR-	32dB at 1MHZ
Power	200μW
Area	800 $(\mu m)^2$

Table 4.4: Opamp Specifications

Although the output can swing between 0.35V and 5V, only the high gain region of approximately 1.5V to 3.5V is used for the signal range. The power and area specifications are for the largest, highest power opamp in the processor. The other opamps were slightly smaller and used slightly less power because they had smaller capacitive loads.

4.4.8 Comparator Design

The comparator schematic is shown in Figure 4.15. The transistor sizes are listed in Table 4.5.

The strobe signal S goes high at the beginning of a comparison cycle. Transistors M12 and M18 place both output stages of the comparator into the same state so that the parasitic capacitors on both sides of the latch circuit will see the same voltage. Transistor M21 turns on and resets the latch by making its outputs equal. The latch consists of transistors M1, M2, M3, and M4, which together form 2 cross-coupled inverters. While S is high, the current source for this latch, transistor M7, is turned off. The comparator inputs are applied to the gates of transistors M5 and M6. If the inputs are different, M5 and M6 will have different currents.

When S goes low, M21 turns off, and the differential current in M5 and M6 causes one of the latch outputs to go high, and the other to go low. (For example, if V_{IN} was higher than V_{REF}, then V_1 goes down while V_2 will go up.) At the same time, M7 turns on and supplies current to M3 and M4. This provides positive feedback which helps push the latch outputs apart. Finally, M12 and M18 turn off while M11 and M17 turn on. The latch outputs are now connected to the comparator outputs through two inverters, which ensure that the comparator outputs swing rail to rail.

Because of the positive feedback introduced by M3 and M4, once the comparator's output is valid, it will not change even if the comparator inputs change. The strobe S must be brought high again in order to perform a new comparison cycle.

When the latch settles to its final value, the node (V_1 or V_2) that goes high may actually reach a higher voltage than the drain of M8. As a result, the current through the input transistor (M5 or M6) reverses direction. For example, if V_{IN} is high then V_2 will go up and reach a voltage that is actually above that of the voltage at the source of M6. At this point, M3 is turned off. So current flows through M7, through M4, then back through M6 to join the current coming from M8. All of this current then flows through M5 and into M1. It is therefore important that the n-well containing M5 and M6 be tied to V_{DD}, *not* the sources of M5 and M6. Otherwise, the PN diode between M6's drain and its n-well can be forward biased.

Name	Size (μm/μm)	Name	Size (μm/μm)	Name	Size(μm/μm)
M1	6.4/0.8	M8	1.6/4	M15	2/0.8
M2	6.4/0.8	M9	2/0.8	M16	1.2/0.8
M3	6.4/0.8	M10	1.2/0.8	M17	1.2/0.8
M4	6.4/0.8	M11	1.2/0.8	M18	1.6/0.8
M5	4.8/2	M12	1.6/0.8	M19	1.6/0.8
M6	4.8/2	M13	1.6/0.8	M20	2.8/0.8
M7	1.6/8	M14	2.8/0.8	M21	3.2/0.8

Table 4.5: Comparator Transistor Sizes

Transistors M7 and M8 are sized to supply currents of 4μA and 7.5μA. When used in the A/D with an opamp as the preamp, a 1LSB differential in the input gives a S falling edge to OUT valid time of about 6ns (in simulation). Sizing of the other transistors is discussed in Section 5.3.

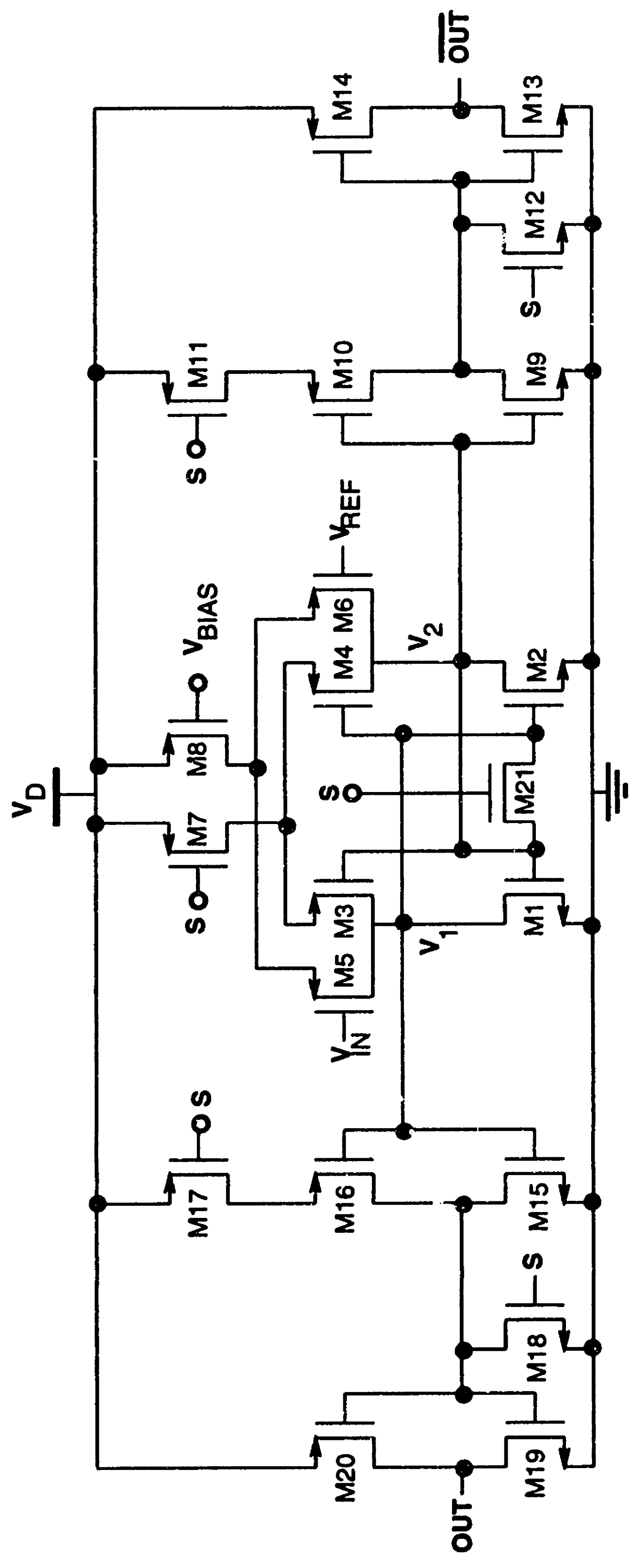


Figure 4.15: Comparator Schematic

4.4.9 ALU Operation

This section describes how the A/D- D/A combination is used to perform the four basic arithmetic operations. It is important to distinguish between voltages and the numerical values which they represent. The signal naming conventions used in this section are shown in Table 4.6.

Name	Description	Range or Value
X_1	Input 1	0 to 256
X_2	Input 2	0 to 256
OUT	Output	0 to 256
V_1	Voltage representing X_1	1.435V to 3.483V
V_2	Voltage representing X_2	1.435V to 3.483V
V_{OUT}	Voltage representing OUT	1.435V to 3.483V
V_G	Voltage representing zero	1.435V
V_{G+K}	Voltage representing K	1.435+K(8mV)
V_{REF}	Voltage representing 128	2.459V
V_{REF2}	Voltage representing 256	3.483V
D_I	Intermediate digital value passed from A/D to D/A	0 to 255
$V_{A/D}$	Voltage that serves as the full scale voltage for the A/D	1.435V to 3.485V
$V_{D/A}$	Voltage that serves as the full scale voltage for the D/A	1.435V to 3.485V

Table 4.6: Signal Naming Conventions

For this circuit, one LSB is 8mV. The zero value is taken to be 1.435V instead of 0V in order to place the input range in the same voltage range for which the opamp has the highest gain. As a result, the midscale is set around 2.5V, and the voltages representing arithmetic values have an offset of 1.435V. Thus, 1.435V represents zero, 1.443V represents one, 1.451V represents two, and so on.

The next four sections describe how the A/D and D/A are configured to implement the various arithmetic operations. In each case, the circuit is configured using NMOS transistors as switches to route the proper signals to the proper inputs. The gates of these NMOS transistors are controlled by the clocks and by 4 bits of the control register.

Addition

The addition operation is implemented in the two extra cycles prior to the 8 cycles of the A/D operation. This is shown in Figure 4.16. Instead of charging C_1 and C_2 to an input V_{IN} during ϕ_1, C_1 is connected to V_1, and C_2 is connected to V_2. Thus, at the end of ϕ_2, the output of the opamp is equal to $(V_1 - V_G) + (V_2 - V_G) + V_G$. This is then passed through the rest of the A/D and D/A operations using V_{REF2} as the reference voltage for both the A/D and the D/A. This gives the result:

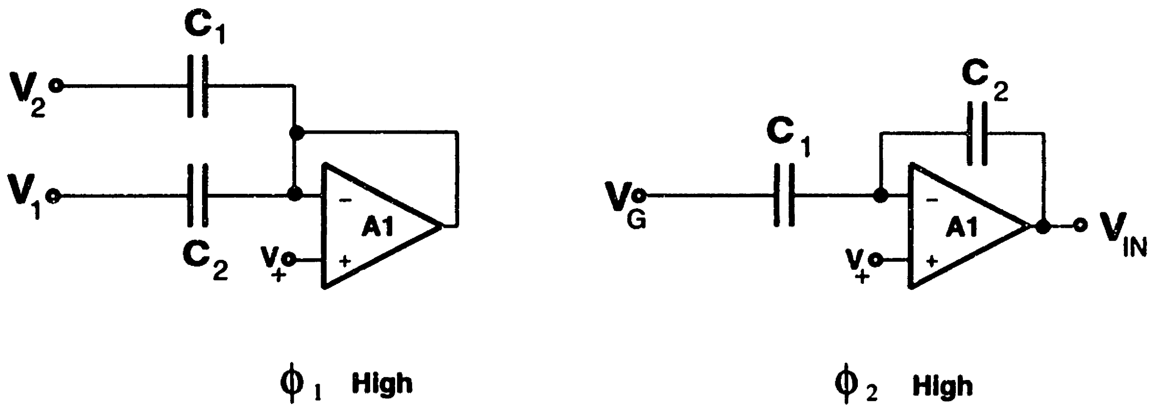


Figure 4.16: Addition Operation

$$V_{OUT} = ((V_1 - V_G) + (V_2 - V_G))\frac{V_{REF2}}{V_{REF2}} + V_G \tag{4.26}$$

$$OUT = MIN(X_1 + X_2, 255) \tag{4.27}$$

This performs the addition. Since the output of the A/D and the D/A can never go above 255, *OUT* will never go above 255.

Subtraction

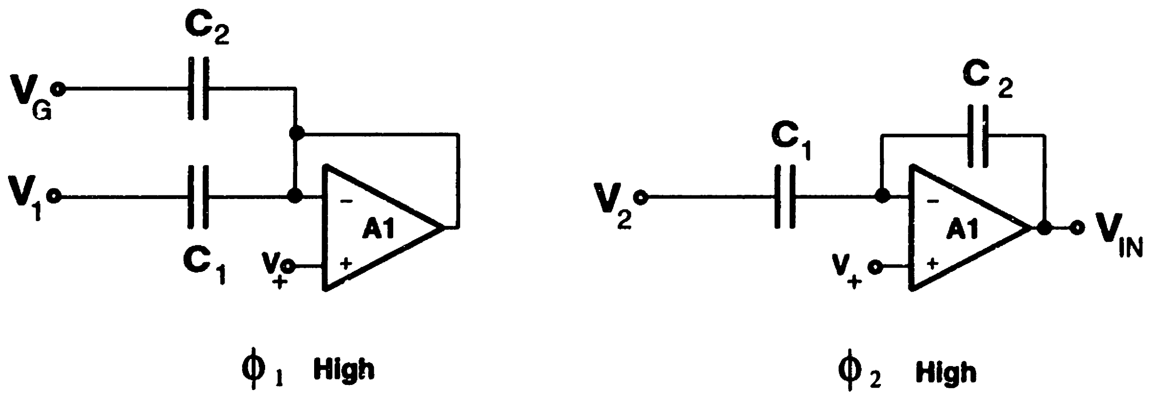


Figure 4.17: Subtraction

The subtraction operation is implemented in the two extra cycles prior to the 8 cycles of the A/D operation. This is shown in Figure 4.17. Instead of charging C_1 and C_2 to an input voltage of V_{IN} during ϕ_1, C_1 is connected to V_1, and C_2 is connected to V_G. Then, during ϕ_2, C_1 is connected to V_2 instead of to V_G. At the end of ϕ_2, the output of the opamp is equal to $V_1 - V_2 + V_G$. This is then passed through the rest of the A/D and D/A operations using V_{REF2} as the reference voltage for both the A/D and the D/A. This gives the result:

$$V_{OUT} = (V_1 - V_2)\frac{V_{REF2}}{V_{REF2}} + V_G \tag{4.28}$$

$$OUT = MAX(X_1 - X_2, 0) \tag{4.29}$$

This performs the subtraction. Since the output of the A/D and the D/A can never go below zero, OUT can never be less than zero.

Multiplication

Multiplication is implemented as shown in Figure 4.18. The input of the A/D is V_1, the reference voltage for the A/D is V_{REF2} and the reference voltage for the D/A is V_2. Thus, using Equation 4.4:

$$V_{OUT} = (V_1 - V_G)\frac{V_2 - V_G}{V_{REF2} - V_G} + V_G \tag{4.30}$$

$$OUT = \frac{X_1 X_2}{256} \tag{4.31}$$

The input V_2 is held by one of the input SH circuits during the A/D operation. It is then transferred to one of the output SH circuits for use during the D/A operation.

Due to the division by 256 in Equation 4.31, the output of the multiplication operation will never exceed its allowed maximum of 256. Also, multiplying any number by zero will result in the correct output of zero.

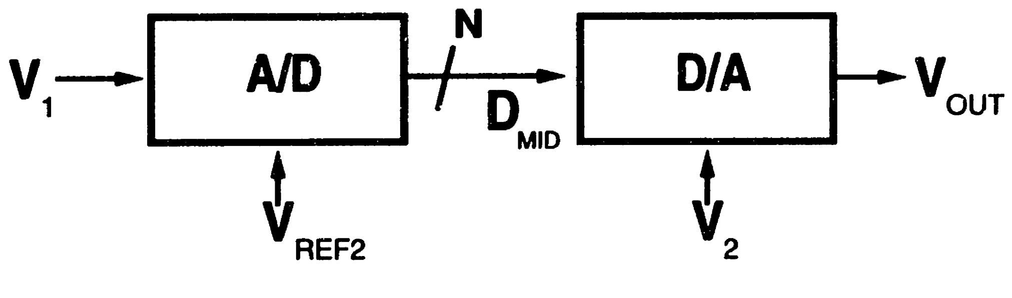


Figure 4.18: Multiplication

Division

Division is implemented as shown in Figure 4.19. The input of the A/D is V_{G+K}, the reference voltage for the A/D is V_1 and the reference voltage for the D/A is V_2. Thus, using

Equation 4.4:

$$V_{OUT} = \frac{V_2 - V_G}{V_1 - V_G} 8KmV + V_G \quad (4.32)$$

$$OUT = MIN(K\frac{X_2}{X_1}, X_2) \quad (4.33)$$

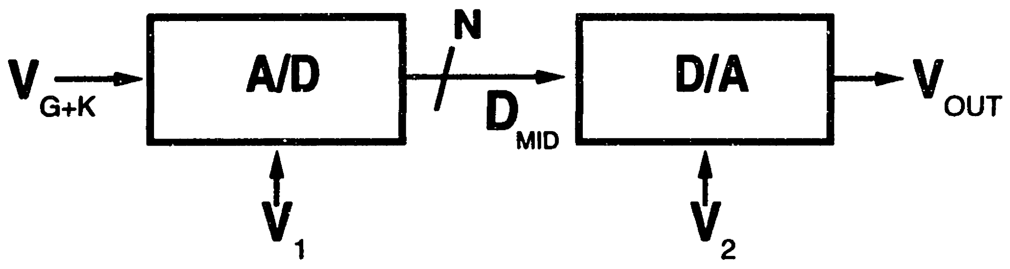


Figure 4.19: Division

Due to the fact that the output of the D/A, V_{OUT}, can never be greater than $V_{D/A}$, the output OUT can never exceed X_2. The factor of K can easily be changed on a chip-wide basis by changing the bias voltage V_{G+K}. Different algorithms will need different values of K.

Examination of the A/D circuit reveals the fact that both V_1 (used as V_{REF2}) and $V_1/2$ (used as V_{REF}) are needed for the A/D's operation. $V_1/2$ is produced during the first two clock phases as shown in Figure 4.20. When ϕ_1 is high, C_1 is connected to V_1, and C_2 is connected to V_G. Then, when ϕ_2 is high, C_1 and C_2 are connected to the output of A1. At the end of ϕ_2, the output of the opamp is equal to $V_1/2$.

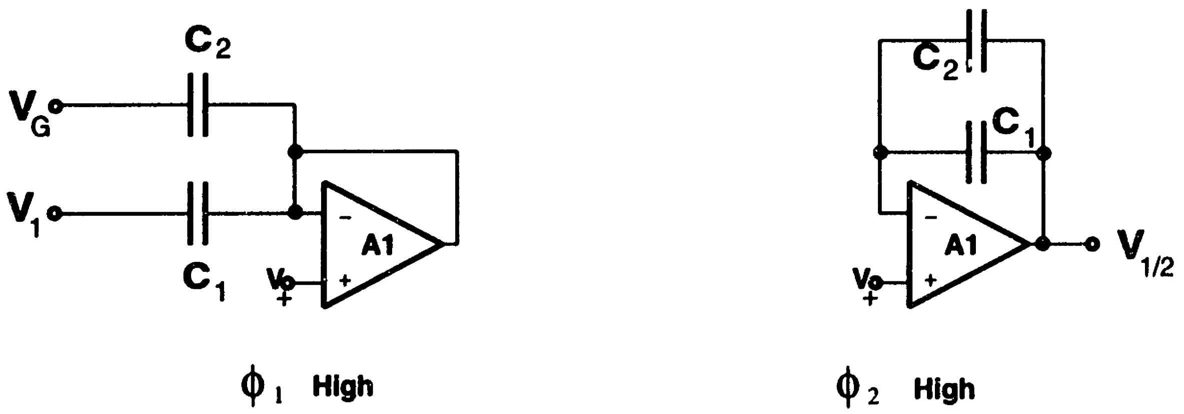


Figure 4.20: Producing $V_1/2$

4.5 Control Register Design

The cell control register is a 35 bit shift register. Its output bits control the 31 pass transistors of the switch fabric and the 4 control bits of the ALU. The control registers of the various processors are connected together in one long line, so that it is essentially a 35P bit shift register, where P is the number of ADAP processors on the chip. The input of the control register is connected to the output of the control register of the previous cell, and the output of the control register is connected to the input of the control register of the next cell. The schematic of one control register is shown in Figure 4.21. The transistor sizes for the control register are shown in Table 4.7.

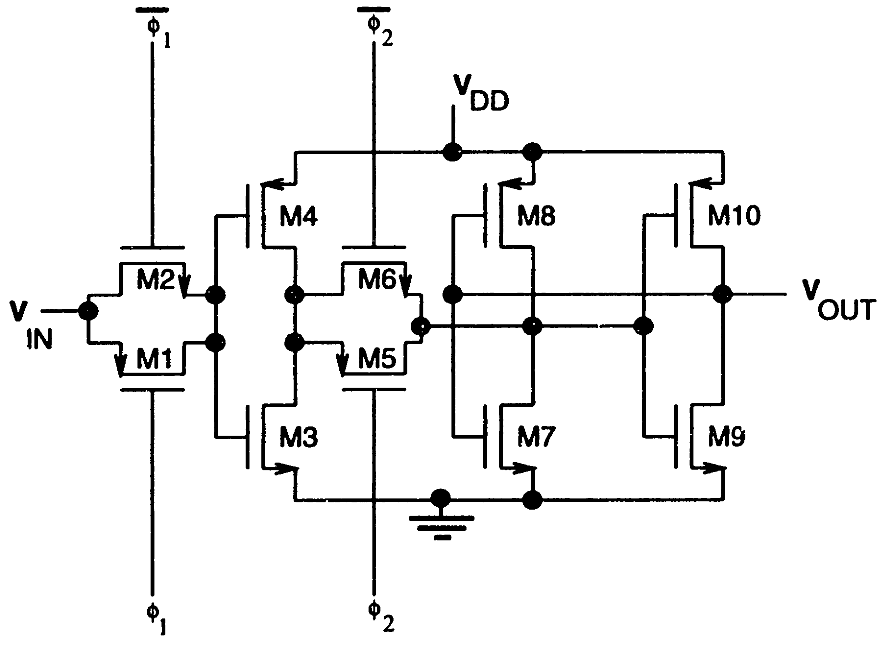


Figure 4.21: Schematic of Control Register

Name	Size (μm/μm)	Name	Size (μm/μm)
M1	1.6/0.8	M6	1.6/0.8
M2	1.6/0.8	M7	1.2/4.6
M3	1.6/0.8	M8	1.2/4.6
M4	2.8/0.8	M9	1.6/0.8
M5	1.6/0.8	M10	1.6/0.8

Table 4.7: Control Register Transistor Sizes

The clocks for the control register are not the system ϕ_1 and ϕ_2. Instead, each cell has clock control logic which creates the clock signals for the control register using the formulas:

$$\phi_{1,CR} = \overline{LOAD\overline{\phi_1}} \tag{4.34}$$

$$\phi_{2,CR} = LOAD\phi_2 \tag{4.35}$$

This causes the control registers to shift data when the LOAD signal is high, and to hold the data when the LOAD signal is low. This method saves power because the control registers are only clocked when necessary, and it saves area because the control registers do not need a special load/noload circuit.

Chapter 5

Circuit Layout

5.1 Overall Layout Strategy

An important goal of ADAP's design is to minimize area. This is the driving force behind much of the circuit design and layout. The circuits were squeezed together by hand as much as possible, and minimum size transistors and minimum wire widths were used whenever possible. In addition, care was taken during the circuit layout to separate the digital and analog circuitry as much as possible. This is done to minimize noise in the analog data from the noisy digital circuits.

The layout of the ADAP chip was performed using Cadence CAD tools, including DIVA for design rule checking, Virtuoso for layout, and Composer for schematic entry.

5.2 Capacitors

The digital CMOS process used for ADAP does not have two layers of polysilicon, so capacitors for the sample and hold circuits are formed using a four layer sandwich of polysilicon (Poly), metal one (M1), metal two (M2), and metal three (M3), as shown in Figure 5.1. Using these four layers increases the capacitance per area and thus decreases the total capacitor area.

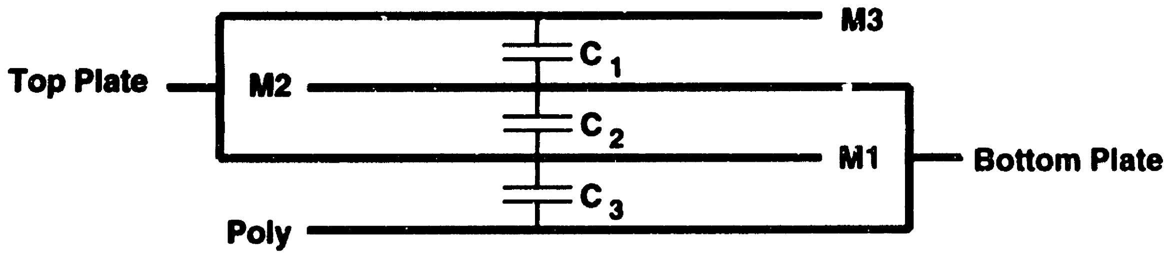


Figure 5.1: Capacitor Structure, $C_{Total} = C_1 + C_2 + C_3$

For the capacitors for the A/D and D/A, it is also important for the capacitors to match each other. Any capacitor mismatch degrades the resolution of the A/D or D/A. Capacitor matching is improved in several ways. First, the capacitors are placed next to each other to minimize variations in the dielectric (silicon oxide) thickness. Next, the four corners are made with two 45 degree cuts instead of 90 degree angles.[1] Also, the capacitors have power or ground rails on all sides so that both of the capacitors have the same fringing fields. Finally, the M3 layer is not used for the capacitors requiring matching. This means that the capacitance is defined entirely by the area of the M1 plate. The M2 and Poly plates are extended about 2μm beyond the M1 plate so that any normal alignment error of M1 does not affect the capacitance. This has two additional benefits. First, the M3 plate has a capacitance to the outside of the chip and thus can couple noise into the inverting input of the opamp, which is very undesirable. Second, the capacitors are sufficiently small (about 53fF), that the area used to connect the M3 and M1 plates in a symmetric manner uses more area than is saved by using the M2 to M3 capacitance.

5.3 Comparator

The important feature of the comparator's layout is that everything must be symmetric. Any mismatch in the two sides of the comparator introduces comparator offset, which degrades the A/D's resolution. The comparator layout is shown in Figure 5.2.

Several layout techniques are utilized to reduce the comparator offset. First, the transistor pairs which need to match are constructed with 8 separate transistors in a common-centroid pattern so that threshold voltage mismatches are canceled [52]. Using large transistors would improve the matching, but it was important to minimize the area, so minimum size transistors are used. In addition, all of the wiring for the comparator is done symmetrically so that the parasitic capacitances on both sides of the comparator match. This is important since the comparator is a dynamic circuit, and any mismatch in parasitics introduces a bias in favor of one output over another.

Tests of comparators on ten different chips give a maximum offset of 31mV. This is acceptable since the opamp used as a preamp for the comparator has a gain of about 3,000. Therefore, the input referred offset is only 10μV, and it should be able to resolve a 16mV LSB easily.

5.4 Analog and Digital Sections

The layout of the ADAP cell is shown in Figure 5.3. The analog power and biases run along the right side of the cell while the digital power and clocks run along the left side of the cell. The opamps and capacitors are placed along the bottom and right sides of the cell while the

[1]The reason why this is done is to increase the area to perimeter ratio. Since most of the matching errors occur at the perimeter, it is desirable to minimize the perimeter for a given capacitor area. However, care needs to be taken that the mask production step can handle the 45 degree angle correctly. Some methods will produce jagged edges which will have worse matching than a simple 90 degree angle.

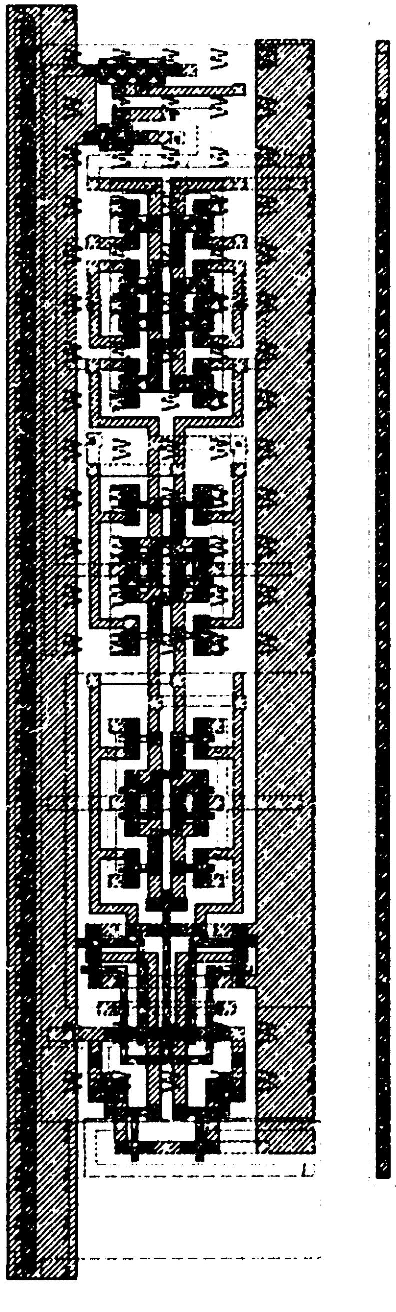


Figure 5.2: Comparator Layout

digital gates and registers are placed along the top and left sides of the cell. The NMOS pass gates which are used to route the analog signals are placed between the digital and analog sections. In these sections, overlap of the digital clock lines and analog data lines is avoided whenever possible.

There is one digital clock signal which extends into the opamps near their inverting node.

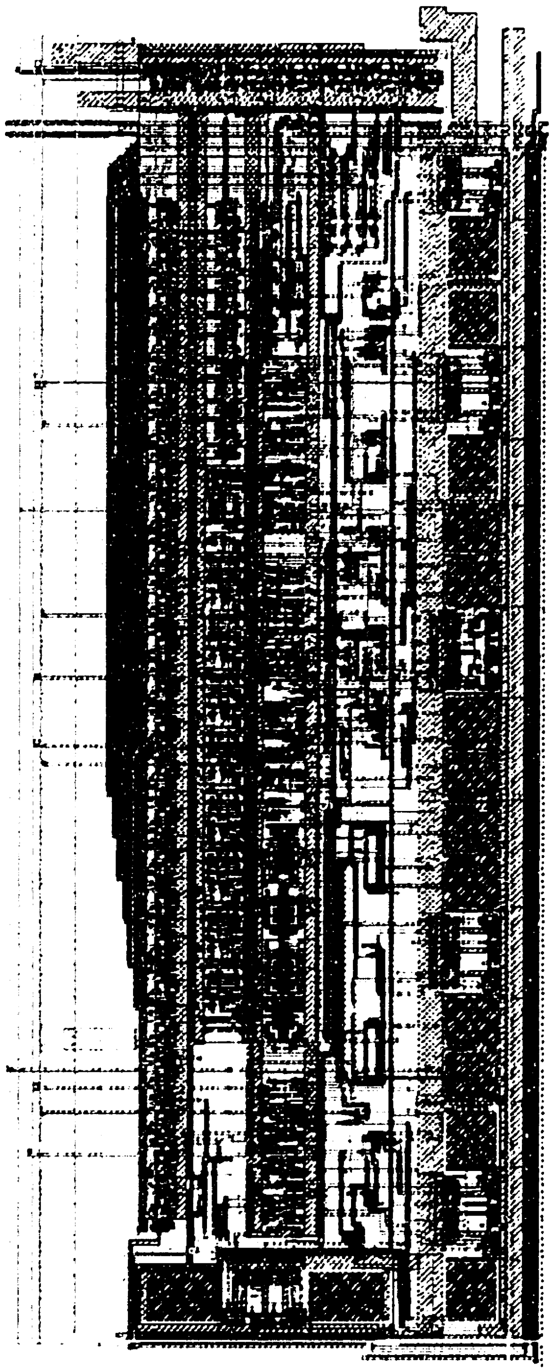


Figure 5.3: ADAP Cell Layout

This clock signal is surrounded by metal tied to the power or ground lines in order to shield the inverting node from the clock signal.

The p-type substrate is tied to the analog ground in a ring around the analog circuits in order to minimize any digital signal passing through the substrate to the analog circuits. The substrate is also tied to analog ground in a ring around all of the capacitors to minimize keep the substrate quiet. Also, when an analog data line crosses the digital clock lines, the data line, which is constructed with M1, is shielded from the clock line, which is carried by M3, by a M2 shield tied to analog ground.

5.5 Bonding Pads

All of the digital pins were placed on one side of the chip, and the analog pins were placed on the other side of the chip. The pin locations are given in Appendix D.

To ensure clean edges, all of the digital inputs (both clocks and data) to the chip are buffered by two inverters in the pad before the signals reach the array. The digital outputs from the array are buffered by 3 graded inverters in the pad before going outside of the chip.

The chip was fabricated without ESD protection in the pads in order to remove any restriction on the possible input voltage range for the pads. Extra power and ground pins were used to reduce resistance and inductance in the power leads.

5.6 Layout Verification

The layout was verified using Cadence's Layout Versus Schematic (LVS) program. This ensured that the layout matched the schematic. In addition, a schematic was extracted from the final chip layout and simulated. Due to the size of the circuit, the simulation was very slow, and only the power-up simulation was able to be performed for the entire array chip. However, since the entire chip was simply an array of cells, extensive simulation of one cell by itself provided a high confidence level for the chip as a whole.

Circuit	Area (μm^2)
ADAP Processor Cell	189k (270μm x 700μm)
A/D	24k
D/A	14k
Input S&H Circuits	20k
Output S&H Circuits	8k
Storage Unit S&H	8k
Digital Buffer	9k
Instruction Register	30k
Switch Fabric	20k
Power Buses, Wiring	56k

Table 5.1: ADAP Circuit Area

5.7 Die Photos

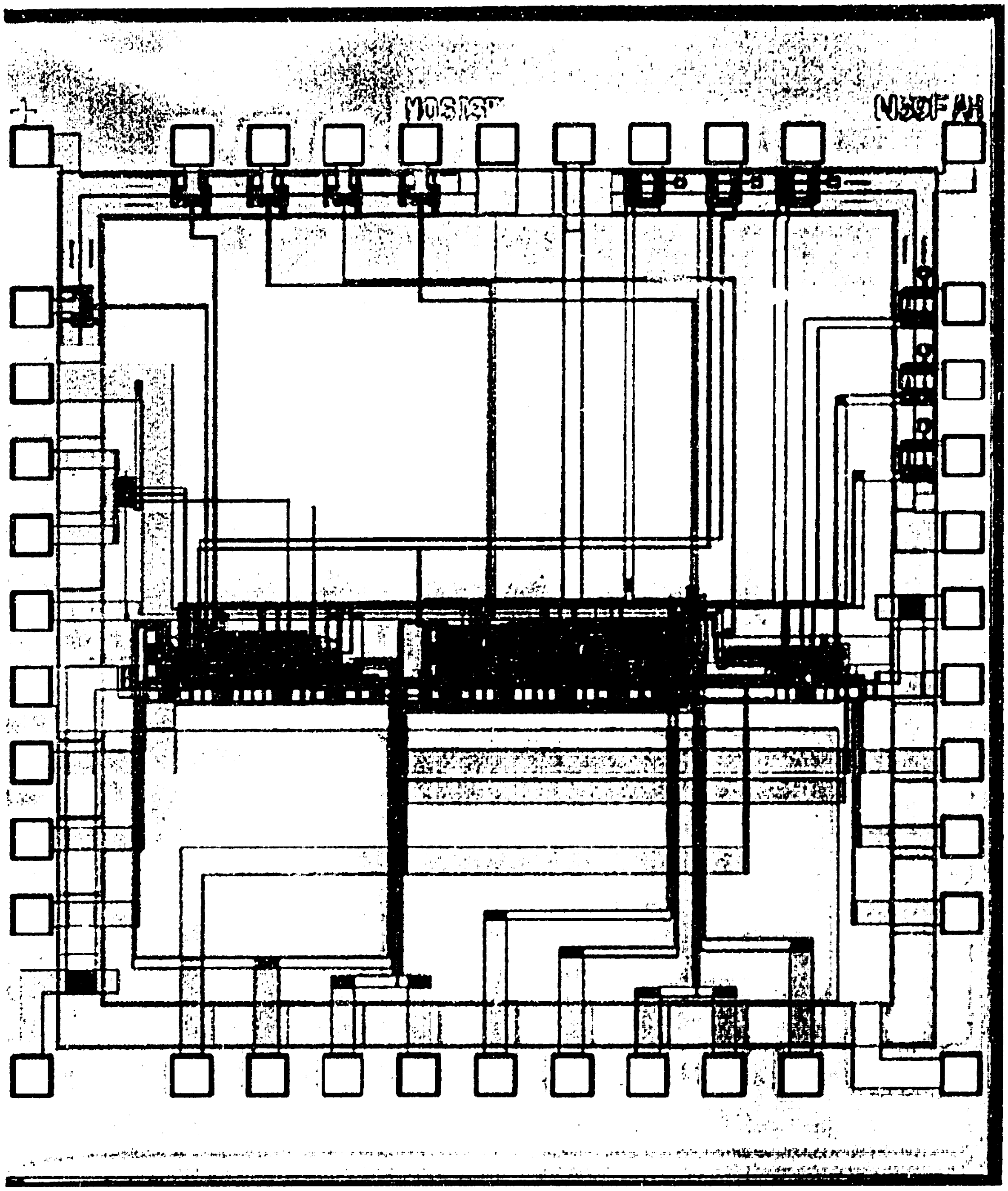


Figure 5.4: ADAP Processor Cell Chip Die Photo

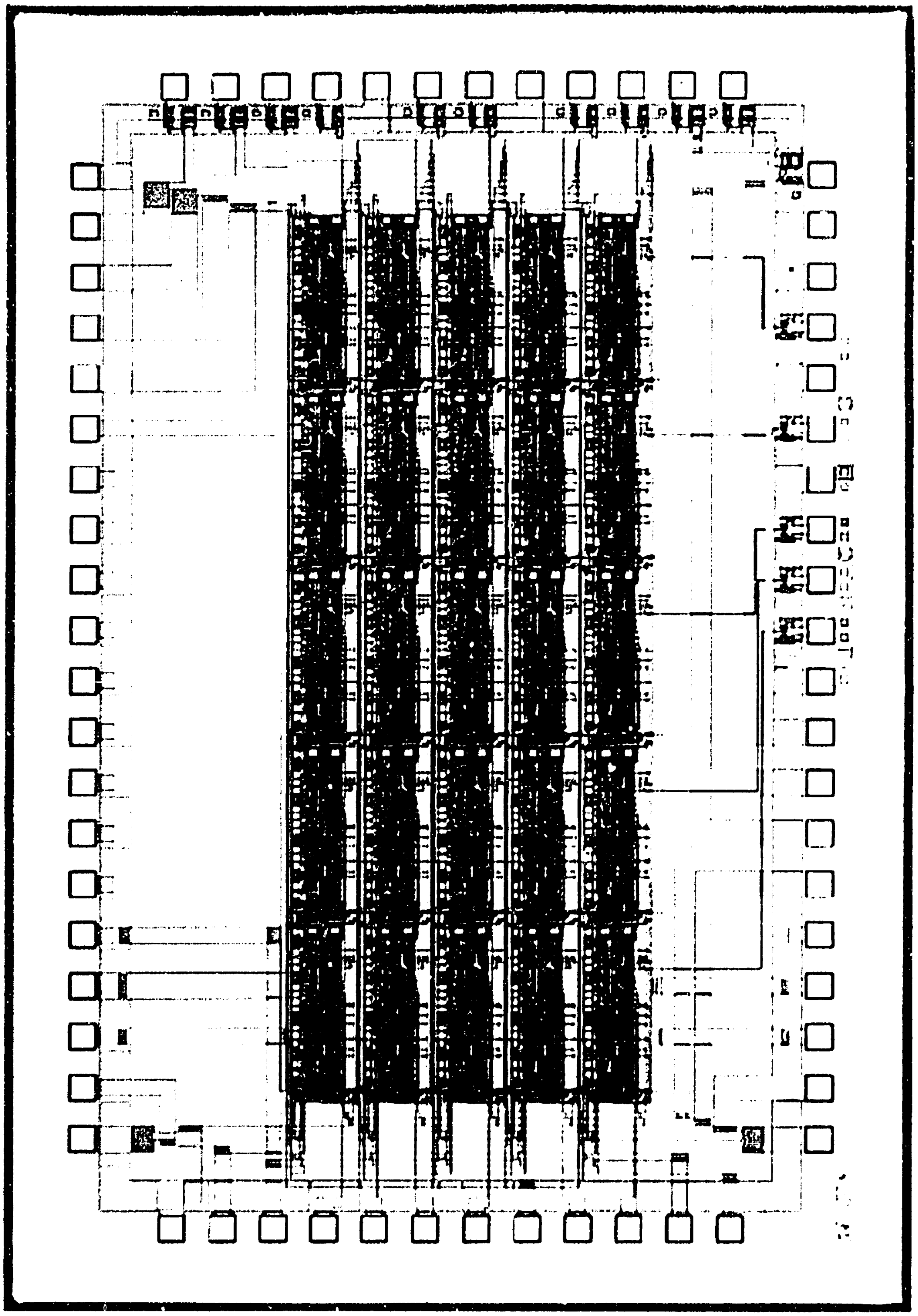


Figure 5.5: ADAP Array Chip Die Photo

Chapter 6

Test Setup

6.1 Overall Test Strategy

Two different chips were fabricated to test the ADAP processor: a processor cell test chip and an array test chip. The cell test chip has copies of one complete processor cell, an ALU, a sample and hold, and the A/D on it. This allows the performance of the various ADAP cell components to be tested directly. The array test chip has a five by five array of cells. This allows the array to be tested with actual vision algorithms. The pinouts for the array chip can be found in Appendix D.

6.2 Test System

A photograph of the test system is shown in Figure 6.1, and a block diagram is shown in Figure 6.2. A Printed Circuit Board (PCB) holds the test chip and interface circuits. A Clock Generator Board generates the digital clock signals and sends them to the PCB via a ribbon cable. The test setup is controlled by a PC. This is connected to a PXB-721 I/O card which is connected to the PCB through a Digital Acquisition Board. Analog inputs from a Data Precision 8200 or an Audio Precision One can be connected to the PCB. The PC sends its data via an Ethernet connection to UNIX workstations for processing and analysis. There are two PCBs. One is used for the cell test chip, the other for the array test chip. Many aspects of the test setup, both software and hardware, are based on the test setup described in [53].

6.3 PCB

The PCB design is shown in Figure 6.3. A five layer board design was used. The three internal layers are used for ground and power planes while the upper and lower layers are used for signal routing. This allows easy modification of the board (by scraping away a signal line) if any design errors are found after fabrication. All signal routes are 30 mil wide lines.

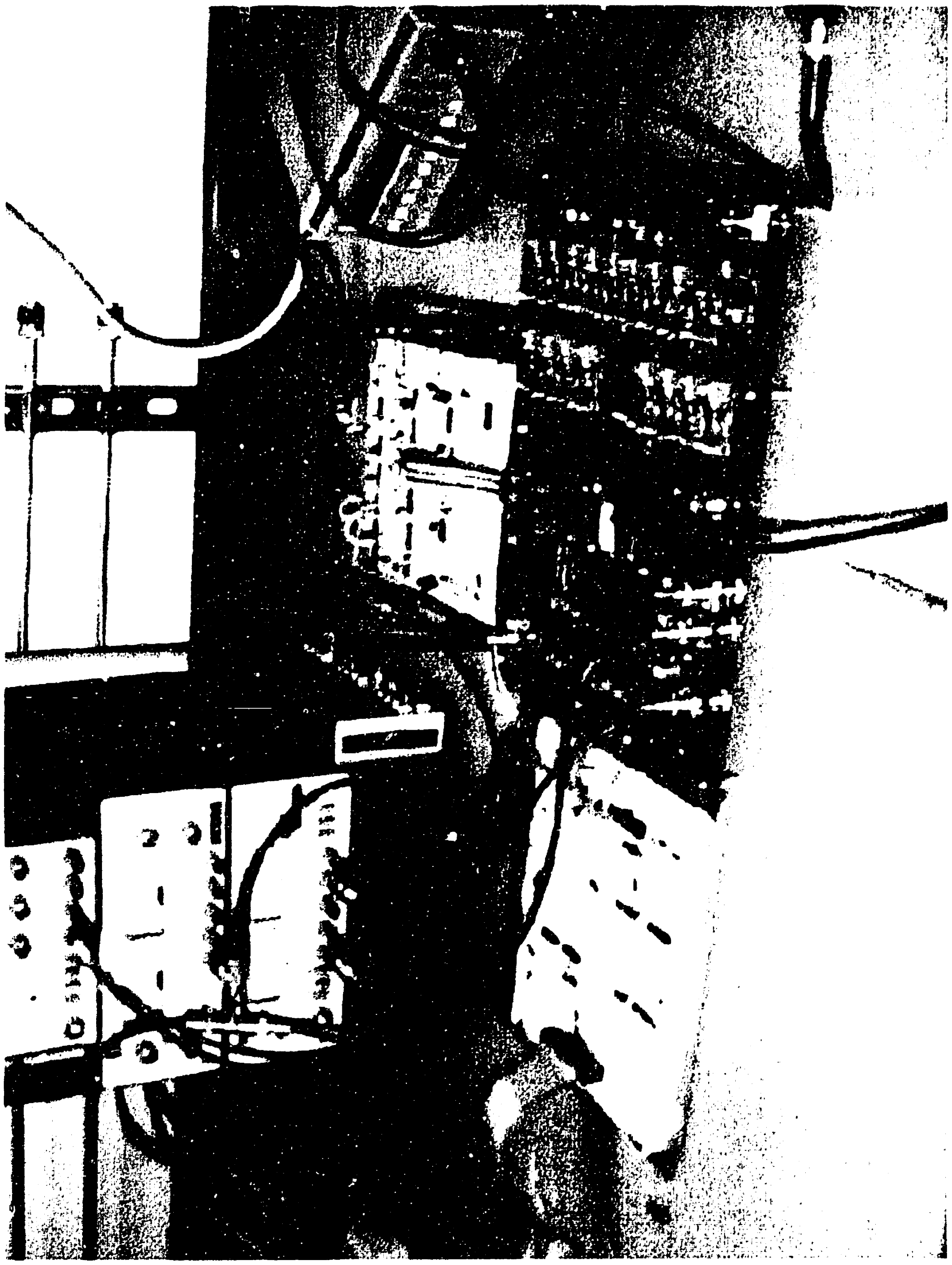


Figure 6.1: Picture of ADAP Test Setup

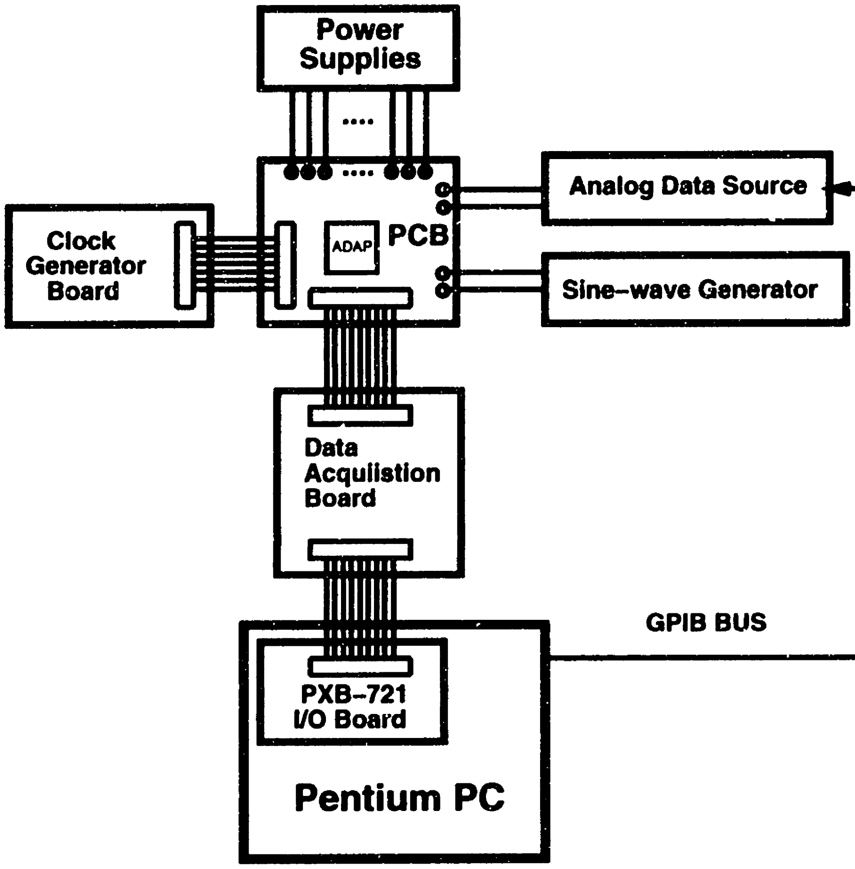


Figure 6.2: Diagram of ADAP Test Setup

The board is designed into four sections: the analog inputs and bias voltages, the internal digital signals, the external digital signals, and the test chip. These are described in detail in the following sections.

6.3.1 Test Chip

The test chip sits in a 65 pin LIF socket soldered directly into the board. This allowed test chips to be inserted and removed without excessive force. $10\mu F$ tantalum and $.33\mu F$ surface mount ceramic capacitors are placed next to the test chip power pins.

6.3.2 Digital Signals

The digital signals were brought on and off the PCB using Burr-Brown ISO150 capacitive digital isolator chips. The ISO150 has two separate power supplies; one for its inputs and one for its outputs. These chips were the only digital chips on the PCB. For the test chip's digital inputs, the ISO150 inputs were connected to the outputs of the clock generator board, and the ISO150 outputs were connected to the digital inputs of the test chip. For the test

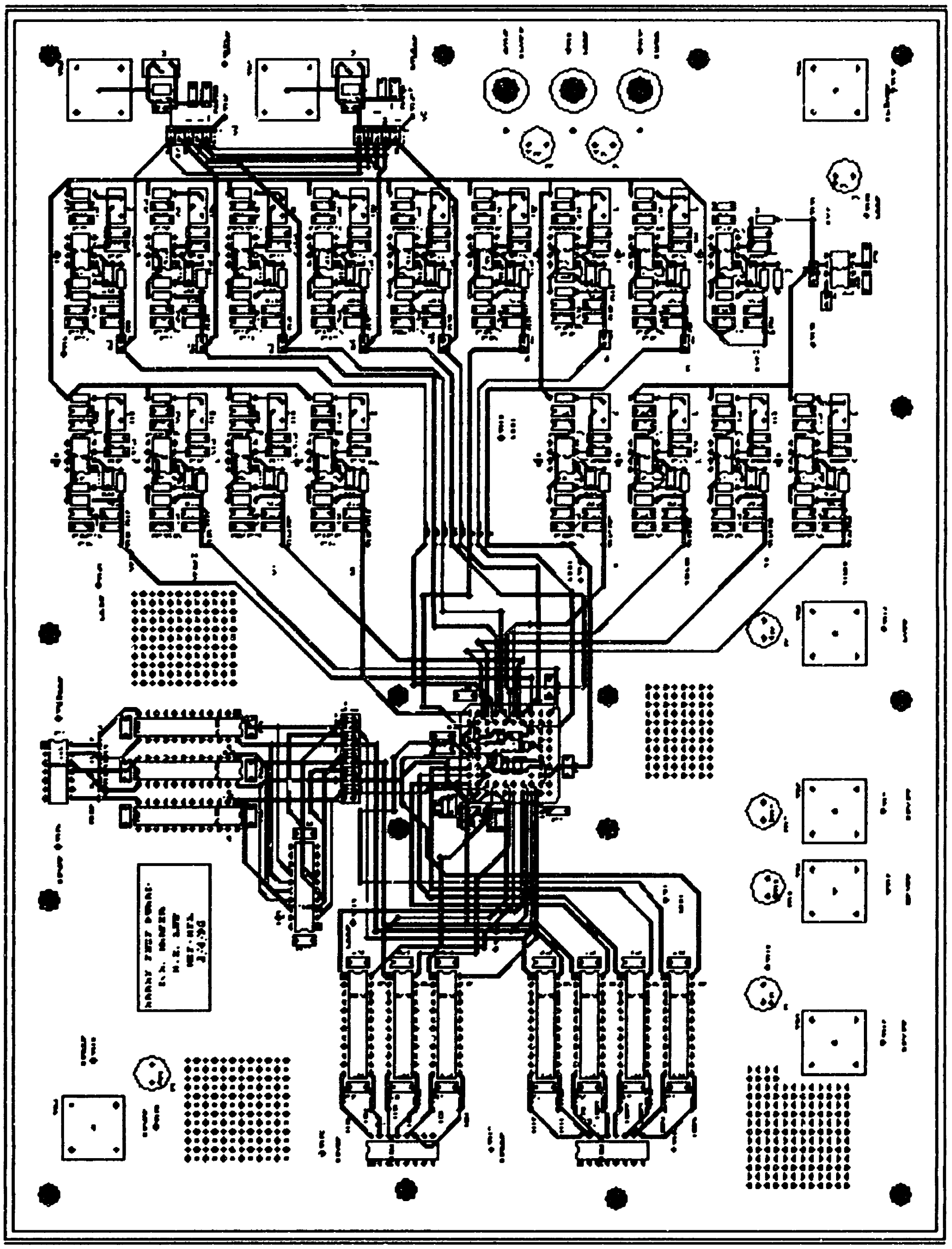


Figure 6.3: Layout of PCB for the Array Test Chip

chip's digital outputs, the ISO150 inputs were connected to the test chip outputs, and the ISO150 outputs were connected to the digital acquisition board. This allowed the digital signals to be generated and processed on other boards without injecting digital transients into the analog power supply and bias voltages. The ISO150 chips also provided TTL to

CMOS and CMOS to TTL level shifting. The ISO150 chips were bypassed with ceramic capacitors.

6.3.3 Analog Bias Voltages

The analog bias voltages and inputs are generated using an AD780 voltage reference. This chip produces a low noise voltage V_{REF} that is distributed around the analog section of the PCB. Resistive dividers are used to generate the exact voltage needed. This voltage is then buffered by a low-noise voltage buffer. The resistive divider and buffer circuit, based on a design in [54, 55], is shown in Figure 6.4. Nominal values for this circuit are $R1 = 9k\Omega, R3 = 1k\Omega, R4 = 47k\Omega, R5 = 22\Omega, C1 = 10\mu F, C2 = 10nF, C3 = 10\mu F$. $R2$ is a $10k\Omega$ potentiometer. Each OP27 opamp has a +/- 15V power supply bypassed with $10\mu F$ capacitors.

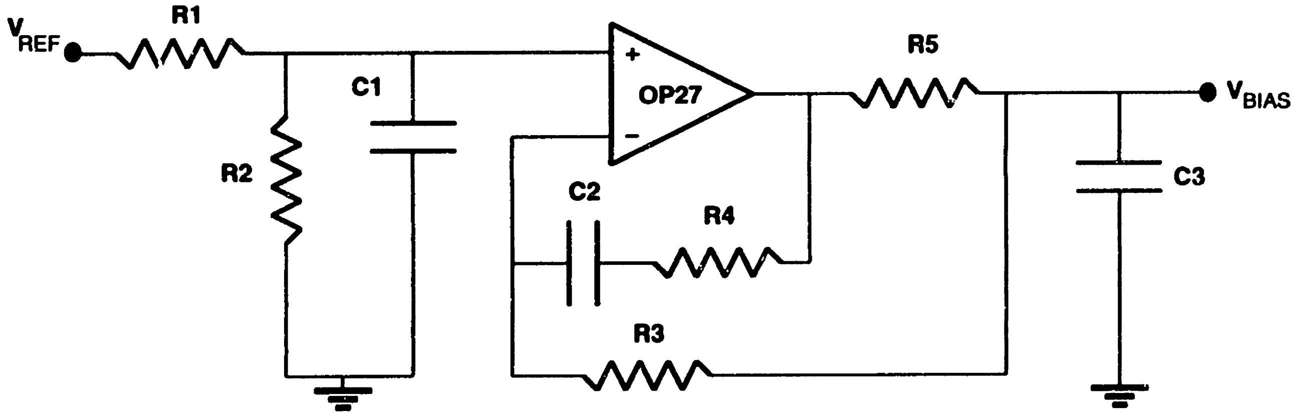


Figure 6.4: Analog Bias Voltage Buffer Circuit

6.3.4 Analog Inputs

Although most analog inputs to the test chip are DC voltages which use the same circuit as the bias voltages, two of the analog inputs can come from off the PCB board as well as from a buffer circuit. The two additional sources used are an Audio Precision System One to provide a sine wave input, and a Data Precision 8200 to provide a digitally controlled DC analog input. The Data Precision is controlled by the PC through a GPIB bus. Both of these inputs are BNC connectors so that shielded BNC cables can be used to connect the signal. A balanced XLR cable is used for the Audio Precision signal to reduce the capacitive load. A small $R_{in}C_{in}$ low-pass filter is placed at the input to the PCB in order to reduce any noise from the input source. C_{in} has a nominal value of 560pF, and R_{in} is chosen to place the filter cutoff frequency above that of the input.

6.3.5 Ground Plane

The center layer of the PCB was used for the ground plane. This layer provided a very low impedance path for ground currents. A star-ground approach was used, as shown in Figure 6.5, in order to minimize the effect of digital transients on the analog signals. In the figure, the black regions indicate a continuous ground plane while the white regions indicate cuts in the ground plane. The ground plane for the ISO150 I/O chips that connect to the digital boards (labeled "digital Circuits Connected to External Boards (CGB and DAB)") is a separate ground plane.

6.3.6 Power Supplies

The test setup uses numerous power supplies. There is a +/-15V power supply for the analog buffer circuits, a 5V power supply for the AD780, and a 5V supply for the test chip's analog power pins. There are two 5V power supplies for the test chip's digital power pins. Finally, there are two 5V power supplies for each side of the ISO150s. All power supplies except the +/-15V supply are connected to the PCB using BNC cables to maximize shielding. All power supplies are bypassed with electrolytic capacitors near the BNC connector and with ceramic and tantalum capacitors near the chips they supply. All bypassing is done to the ground plane since all of the signals in the system are referenced to ground.

6.4 Clock Generator Board

The Clock Generator Board (CGB) generates the clock signals ϕ_1, ϕ_2, V_F, V_L, and V_{SHIFT} using TTL logic on a breadboard. Crucial clock delays are trimmed by adding discrete capacitive loads to inverter chains. The clock signals are connected via a ribbon cable to the PCB board. The clock generator board is isolated electrically from the test chip by the ISO150 chips, so the digital noise of the clock generator board does not affect the analog signals on the PCB. The fundamental clock for the clock generator board is supplied by an HP 8116A Pulse/Function Generator. For the FFT test for the A/D, a crystal clock generator chip is used as the fundamental clock source due to its low jitter.

6.5 Digital Acquisition Board

The Digital Acquisition Board (DAB) provides an interface between the PCB and the PC. The DAB is constructed with TTL logic on a breadboard. It performs serial-to-parallel conversion and resynchronizes the digital data for the PC. The DAB is connected to the PCB via two ribbon cables, and the DAB is isolated electrically from the test chip by the ISO150 chips, so the digital noise of the DAB and the PC does not affect the analog signals on the PCB.

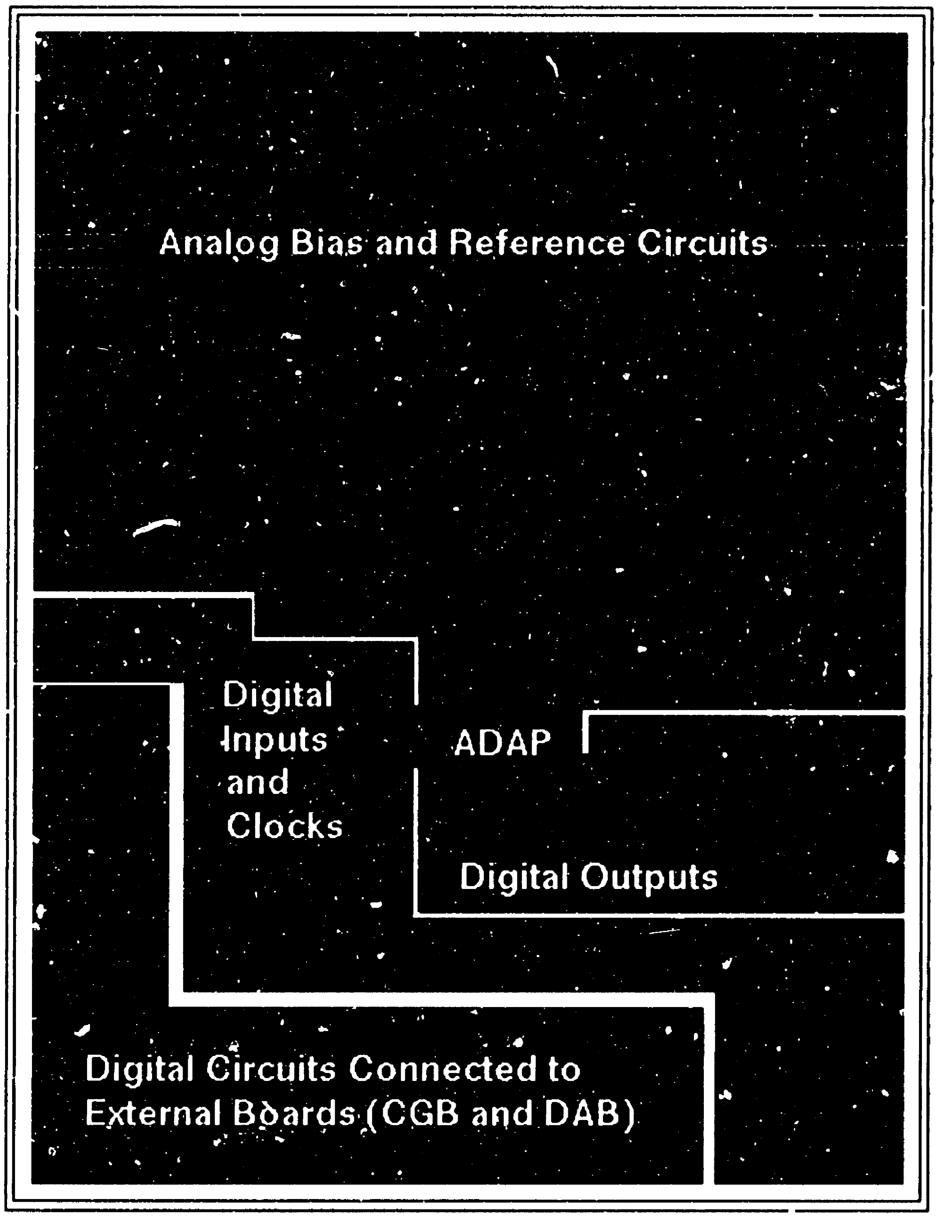

Analog Bias and Reference Circuits
Digital Inputs and Clocks
ADAP
Digital Outputs
Digital Circuits Connected to External Boards (CGB and DAB)

Figure 6.5: Layout of the Ground Plane for the Array Test Chip PCB

Chapter 7

Experimental Results

Two test chips were fabricated and tested. The first was a cell test chip, which contained a processor cell and several stripped down versions of a processor cell. This allowed for independent testing of the A/D, D/A, sample and hold, shift register, control register, and the four arithmetic functions. The array chip contained a 5x5 array which allowed various vision algorithms to be run in order to test the system's functionality.

7.1 A/D

7.1.1 Harmonic Distortion

To measure the harmonic distortion of the A/D, a sine-wave from the Audio Precision System One is fed to the A/D. The digital output of the A/D is then collected. This data is windowed by a Blackman window to reduce the effects of truncating the sine-wave [56]. An FFT is then performed on the data. The signal to noise and distortion ratio (SNDR) is then determined by dividing the power at the input frequency by the power at the other frequencies between DC and the Nyquist frequency. Note that the noise and distortion includes electrical noise from the A/D converter, electrical noise from the test setup, the quantization noise of the A/D converter, and any distortion introduced by the A/D converter. It is possible for the sine-wave generator to introduce distortion, but the Audio Precision's output has distortion of about -100dB, far less than anything measured in these tests [53].

The A/D was measured at a clock rate of 3.33MHz (a conversion rate of 666kS/s). 2,048 points of data were taken at a sampling rate of 41.625kHZ. The sine-wave input frequency was 12.345kHz at an amplitude of $.99V_{Full\ Scale}$. The SNDR is 42.4dB, giving an effective resolution of about 7 bits. The FFT is shown in Figure 7.1.

The A/D was also clocked at 5MHz (a conversion rate of 1MS/s), and 16,384 points of data were taken at a sampling rate of 125kHZ. The sine-wave input frequency was 59.875kHz at an amplitude of $.99V_{Full\ Scale}$. The SNDR is 40.2dB. The FFT is shown in Figure 7.2. Note that, as more datapoints are taken, the quantization noise floor goes down correspondingly [57].

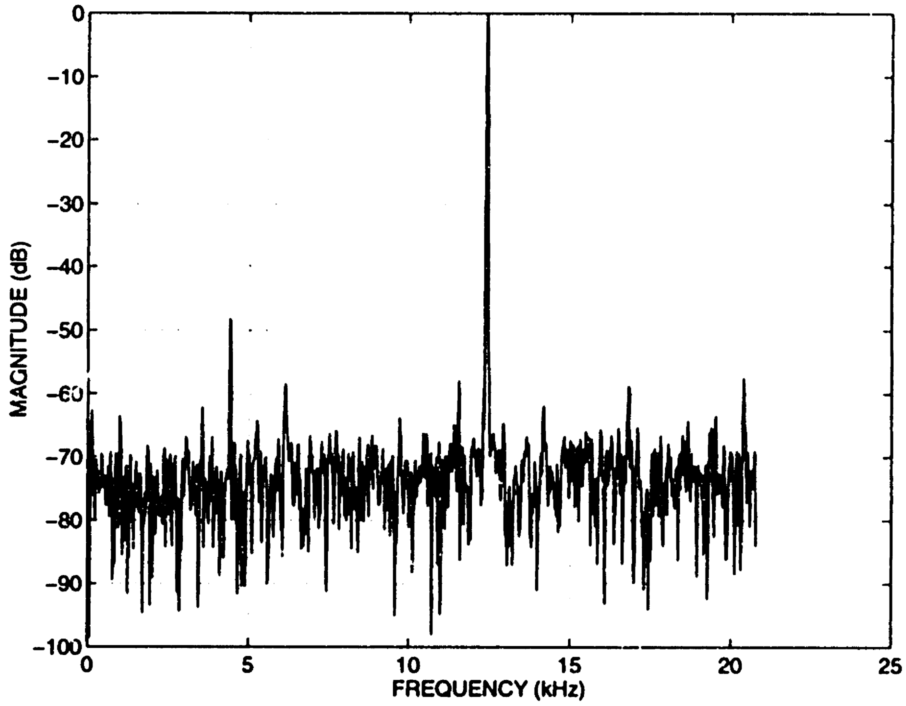


Figure 7.1: FFT Plot, F_{CLK}=3.33MHz, F_{IN}=12.345kHz

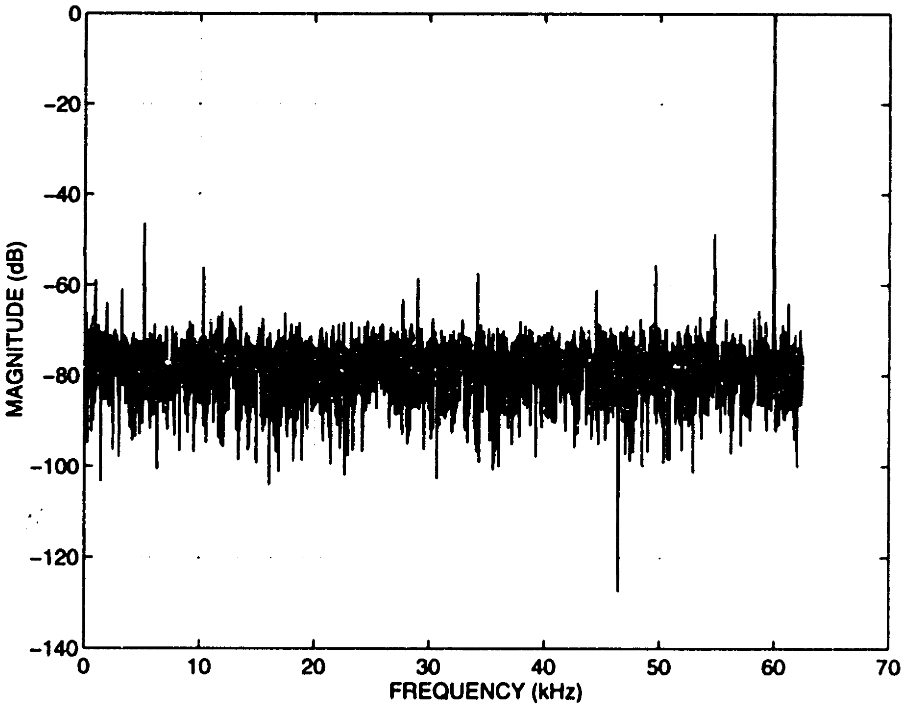


Figure 7.2: FFT Plot, F_{CLK}=5MHz, F_{IN}=59.875kHz

In both FFT plots, the harmonics of the input frequency are above the sampling frequency, but they appear in the plots due to aliasing. In Figure 7.1, the 3rd harmonic is aliased to 4.5kHz, and the 5th harmonic is aliased to 20.1kHz. In Figure 7.2, the 2nd harmonic is aliased to 5.25kHz, the 3rd harmonic is aliased to 54.625kHz, the 5th harmonic is aliased to 49.375kHz, and the 7th harmonic is aliased to 46.74kHz.

7.1.2 INL and DNL

The DNL and INL of the A/D were measured using a sine-wave code density test [56] with a sine-wave input at 12.345kHZ. The input was generated using the Audio Precision System One. The test was run for approximately 2 minutes, and 1 million samples were taken. The A/D was clocked at 5.5MHz (a conversion rate of 1.1MS/s). The A/D was measured at the 7 bit level, where each LSB represents 16mV. Figure 7.4 shows the DNL plot. The peak DNL is -.55LSB. The INL plot is shown in Figure 7.3. The peak INL is -1.16LSB.

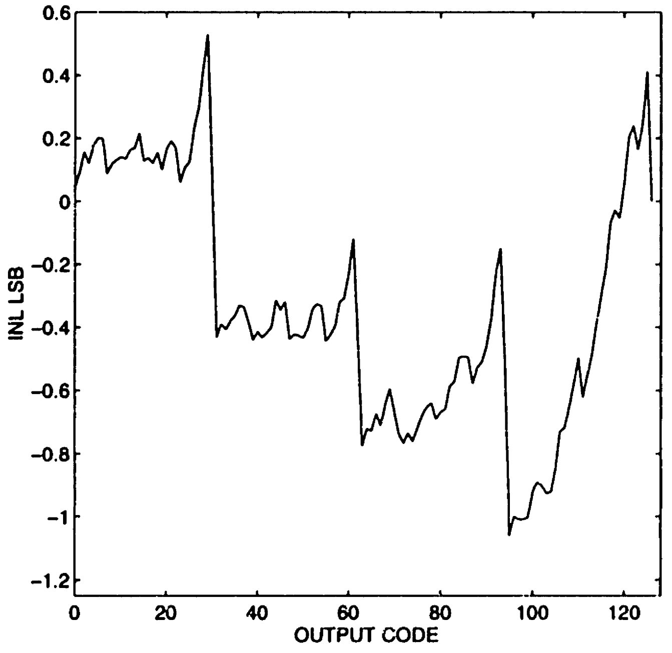


Figure 7.3: A/D INL Plot, F=5.5MHz, LSB is for 7 bits of resolution

One unusual characteristic of the A/D DNL plot in Figure 7.4 is that the peak DNL occurs at the bit 6 (second MSB) transition points, not t`e bit 7 (MSB) transition point. When the A/D is clocked very slowly, at 500kHz, the peak DNL occurs at the bit 7 transition point. This is shown in Figure 7.5. At this slow speed, the main source of error is charge injection and capacitor mismatch, which cause a peak DNL at the bit 7 transition point. When the clock is speeded up, there are additional errors from the opamp settling time as well as possible contamination of the reference voltages, the power supply, and the analog input by the clock. It is difficult to simulate the clock contamination, but the effects of

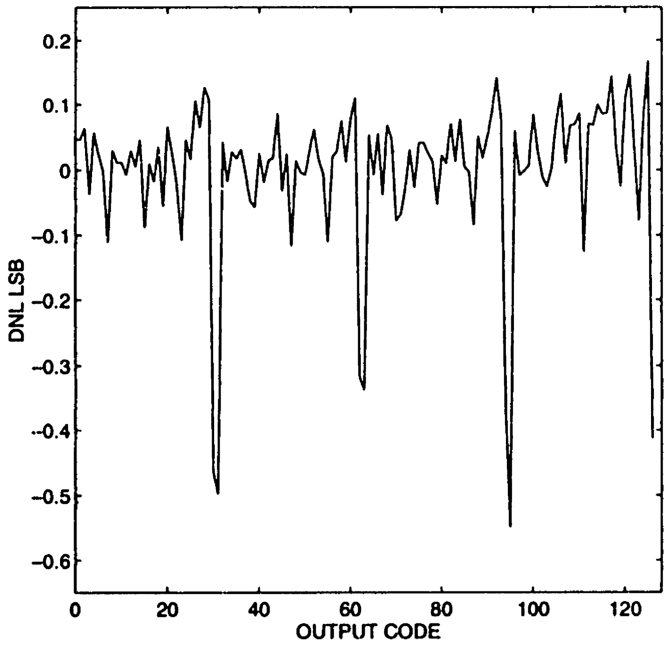


Figure 7.4: A/D DNL Plot, F=5.5MHz, LSB is for 7 bits of resolution

opamp settling time were investigated through simulation. At the bit 7 transition point, the capacitor mismatch and opamp settling time errors partially cancel each other out because the capacitor mismatch produces intermediate voltages that are too high while the opamp's long settling time produces intermediate voltages which are too low. The two effects are not as equal and opposite for the bit 6 transition points, so the cancelation effect is smaller. As a result, the DNL for high clock rates is actually higher at the bit 6 transition points.

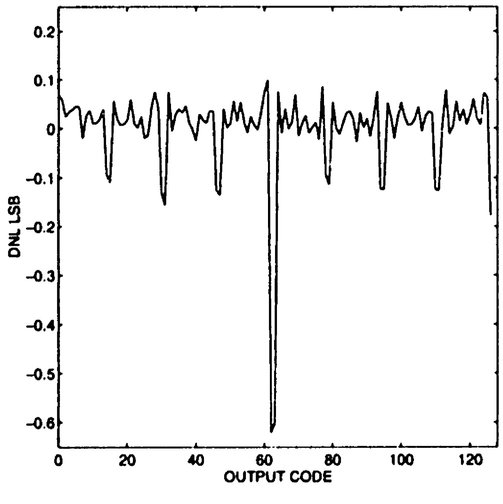


Figure 7.5: A/D DNL Plot, F=500kHz, LSB is for 7 bits of resolution

The A/D was designed to be able to operate at 10MHz. However, its maximum operational speed was about 5.5MHz. One important reason for this involves the clock signals. The actual clock signals that control the switches in the A/D (and the rest of the cell) are not ϕ_1 and ϕ_2 but ϕ_{1D} and ϕ_{2D}. These are delayed versions of the clock signals, as explained in Appendix B.2. The circuit which produces ϕ_{1D} and ϕ_{2D} has a longer delay than simulations indicated. This means that about 25ns of each half cycle is spent just waiting for the clock signal to go high. The reason for the longer-than-simulated delay is that the very large parasitic capacitance between ϕ_{1D} and ϕ_{2D} is not extracted from the layout and is thus not accounted for in simulation. This happens because the Cadence program does not extract parasitic capacitances between two metal lines that are next to each other, only parasitic capacitances between lines that are on top of each other. Figure 7.6 shows how the delay circuit is implemented. Not only is C_{LOAD} not known, but the delay of the circuit is too dependent upon the value of C_{LOAD}. Figure 7.7 shows how the circuit should be implemented. The last inverter should obviously be sized to drive the actual C_{LOAD}, but in this case, the delay is less dependent upon the value of C_{LOAD}. Thus, this circuit has a more predictable delay and the output waveform has sharper edges. In addition to this effect, the various other interline parasitic capacitances probably added to the opamps' total capacitive load, thereby slowing them down even further.

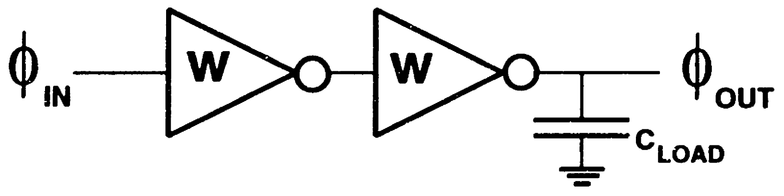


Figure 7.6: Actual Implementation of Delay Circuit (W=weak)

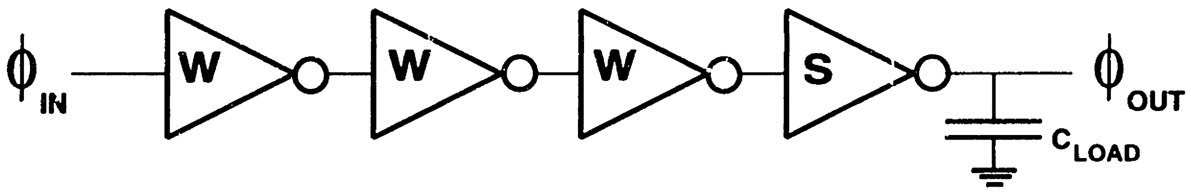


Figure 7.7: Correct Implementation of Delay Circuit(W=Weak, S=Strong)

If the A/D (and D/A) were used as stand-alone circuits, they would not need the extra clock cycle used for the arithmetic functions. Thus, the conversion speed would be 1.375MS/s rather than 1.1MS/s.

7.2 D/A

7.2.1 Harmonic Distortion

The D/A's harmonic distortion was measured at a 5MHz clock speed. A digital 1.000kHz sine wave was sent into the D/A, and the D/A's output was then measured by the Audio Precision's distortion analyzer. This yielded an SNDR of 42.6dB. The actual SNDR may be even better since the D/A output is passed through an output driver, which may introduce a certain amount of distortion. Also, this output driver is disconnected from the chip output for one tenth of the time (during the cycle when it is sampling the D/A's output).

7.2.2 DNL and INL

The DNL and INL of the D/A were also measured. All possible digital inputs were fed into the D/A, and the analog output was measured with a PM2525 multimeter. The D/A was clocked at 5.5MHz (a conversion rate of 1.1MS/s), and it was measured at the 7 bit level. Figure 7.8 shows the DNL plot. The peak DNL is −.77LSB. The INL is shown in Figure 7.9. The peak INL is 1.08LSB.

Since the D/A was not designed to drive analog signals off-chip, it was difficult to measure the analog output with great accuracy. Since the D/A's output is driven off-chip by a sample and hold circuit, the input-output characteristic of the SH circuit was first measured. By inverting this plot, the D/A output prior to being driven off-chip by the SH circuit could be determined.

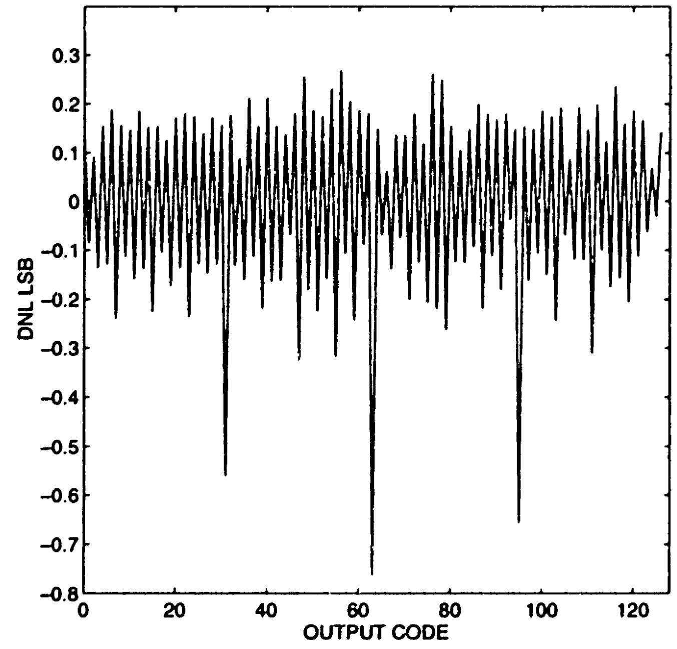


Figure 7.8: D/A DNL Plot, F=5.5MHz, LSB is for 7 bits of resolution

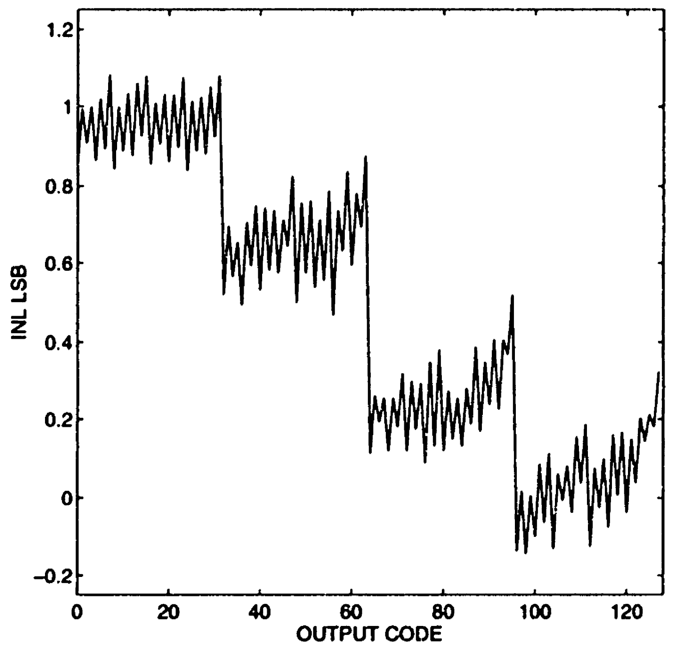


Figure 7.9: D/A INL Plot, F=5.5MHz, LSB is for 7 bits of resolution

Parameter	A/D Measured Value	D/A Measured Value
Resolution	7-bits	7-bits
Conversion Rate	1.1MS/s	1.1MS/s
SNDR	40.2dB	42.6dB
DNL	$< \pm.55$ LSB	$< \pm.77$ LSB
INL	$< \pm1.16$ LSB	$< \pm1.08$ LSB
Analog I/O Range	2V	
Power Supply	0-5V	

Table 7.1: Summary of the A/D and D/A Performance

7.3 Analog Storage Unit

The Analog Storage Unit's sample and hold circuit was tested by simply providing it with an input voltage and then reading the output voltage with a PM2525 multimeter. The I/O plot is shown in Figure 7.10. The input voltage versus error is shown in Figure 7.11. The error is given in LSB at the 7 bit level. (One LSB = 16mV). The peak error is 1LSB at an input voltage of 3.5V.

The circuit's hold time was also measured. The circuit's output degrades with time due to charge leaking off the storage capacitor. It was found that the hold times for 8 bit accuracy (8mV) varied from 4 minutes to 20 seconds in the worst case. Since the maximum hold time needed for most vision applications is 1/30th of a second, the circuit's actual hold time is more than adequate.

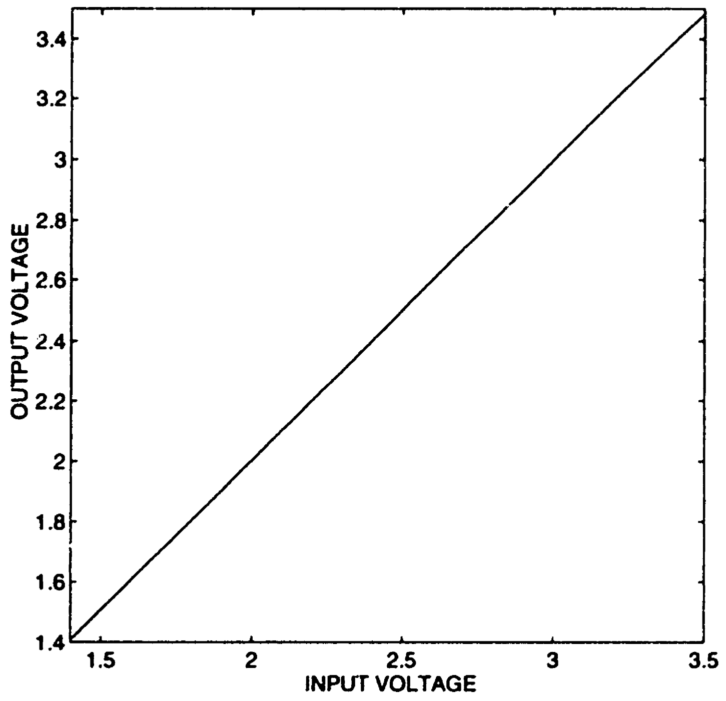


Figure 7.10: S/H Data Plot, Input Versus Output

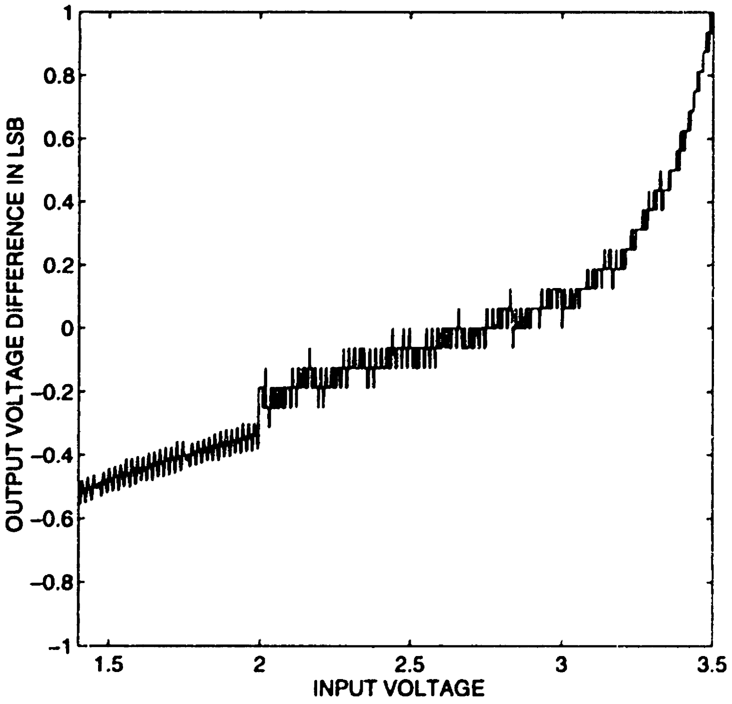


Figure 7.11: SH Data Plot, Input Versus Error, LSB is for 7 bits of resolution

If the sample and hold drives an output pad, it's slew rate goes down due to the increased capacitive load of the pad, PCB traces, and capacitive load of the device connected to the output pin. For the current test setup's parasitic capacitances and a scope probe with a load capacitance of 10pF, the worst case slew rate is .25V/μs.

Finally, the signal degradation as data is passed from one cell to another was measured. This was done using the array chip. An analog voltage was presented to the input of one cell. The signal was then passed through all 25 cells and measured at the output. This test takes into account not just the error of the sample and hold circuit itself but also any settling time errors. As the clock speed is increased, the circuit has less time for its output to settle to its final value. At 5MHz, the worst case error was +8.7mV, or about 1 LSB at the 8 bit level, per cell. The best case error was 4.1mV, or about 1/2 LSB, per cell.

7.4 Arithmetic Functions

The four arithmetic functions were also tested. Each function has two inputs, V_1 and V_2, which can vary from 1.435V to 3.483V, representing 8-bit values from 0 to 256, with 8mV per LSB. Note that the A/D, D/A, and SH data were shown for 7 bits of resolution, but the arithmetic data will be shown for 8 bits of resolution. In each case, V_1 was held constant while V_2 was ramped through all possible values. Then V_2 was held constant while V_1 was ramped.

In addition to V_{IN} versus V_{OUT} data, the accuracy of the arithmetic functions was measured by plotting the difference between the ideal output and the actual output. This is similar to the INL measurement of an A/D; it will be called the absolute error. Also, the average offset between the actual output and the ideal output can be calculated. If this offset is then subtracted from the absolute error, a differential error can be found. This is similar (but not the same) as the DNL of an A/D. This will be called the differential error.

The arithmetic functions were tested at 4MHz. Addition, subtraction, and multiplication worked at a clock speed of 5MHz, but the division operation required a lower clock speed. This is discussed in Section 7.4.4. Since the array can only have one clock, the slowest operation determines the overall clock speed. The results are summarized in Table 7.2.

7.4.1 Addition

The addition I/O plots are shown in Figure 7.12. When the sum of the two inputs exceeds 255, the output saturates at 255 because the D/A cannot produce an output above 255. The I/O plot is not smooth because the A/D quantizes the input. The peak absolute error of the addition function, shown in Figure 7.13, is 2.9LSB. The peak differential error is ±1.8LSB.

7.4.2 Subtraction

The subtraction I/O plots are shown in Figure 7.14. When the output goes below 0, the output saturates at 0 because the D/A cannot produce an output below 0. The I/O plot is not smooth because the A/D quantizes the input. The peak absolute error of the subtraction function, shown in Figure 7.15, is 3.1LSB. The peak differential error is ±2.3LSB.

7.4.3 Multiplication

The multiplication I/O plots are shown in Figure 7.16. The I/O plot for a V_1 ramp input is not smooth because the A/D quantizes the input. However, when the V_2 input is ramped, the output is smooth because it is the D/A reference, which is not quantized, being varied.

The peak absolute error of the multiplication function, shown in Figure 7.17, is 3.5LSB. The peak differential error is ±1.95LSB. The change in the magnitude of the quantization in the plot of the error for a V_2 ramp is due to the fact that the multimeter changes its resolution at that voltage. This can be seen in Figure 7.11 as well.

7.4.4 Division

The division I/O plots are shown in Figure 7.18. V_{G+K} was set to 1.507V, which gives a K of 9. The output for ramping the V_2 input is similar to that of the multiplication function, which is to be expected. The plot for a V_1 ramp input shows that the output saturates at V_2 because the D/A output cannot be higher than its reference, which in the division operation is V_2.

The error for the division operation is much higher than for the other operations at low values of V_1. This is due to the fact that V_1 is used as the reference for the A/D. When V_1 is low, the voltage range of the A/D is reduced, but the charge injection is not reduced. As a result, the resolution of the A/D goes down. Thus, the peak absolute error is −16LSB. However, for values of V_1 above 45, the peak absolute error is 3.4LSB, and the peak differential error is ±1.65LSB.

Another cause for the extra error for the division operation is that in order for the A/D to use V_1 as a reference, the A/D circuit also needs the voltage $V_1/2$. This is produced using the circuit in Figure 4.20. This division operation is not perfect; in simulation, it was accurate to about 9 bits of resolution.

The I/O plot for values of K less than 9 shows a more dramatic error for the low denominator region. Values of K greater than 9 have less error, but they sacrifice output range since the output cannot go below K (for high values of the denominator) or above 255 for numerator values below K.

The division operation is slower than the other operations because it requires a longer non-overlapping period. The other operations can run at 1.0 MIPS (a clock speed of 5MHz), whereas the division operation can only run at 0.8MIPS (a clock speed of 4MHz). The reason why this is so can be seen by examining Table B.1. During the first cycle of a division operation, when ϕ_1 is high for the first time, V_1 is sampled. During the next cycle, when ϕ_2 is high for the first time, V_1 is divided by two. When ϕ_2 goes low, the output of ISH1, which has been sampling $V_1/2$, must settle out to $V_1/2$. It must do this at the same time that its output is driving the bottom plates of two capacitors. After its output settles out, the opamp output must slew up or down enough to overcome any comparator offset. All of this must be done by the end of the non-overlapping period. As a result, all of the non-overlapping periods must be long enough for this to happen. One way to speed up the operation is to use a clock signal in which only the second non-overlapping period is long. Producing such a clock signal can be very difficult.

Function	Maximum Absolute Error		Differential Error Range	
	LSB	%	LSB	%
Addition	2.9	1.1	± 1.8	± .7
Subtraction	3.1	1.2	± 2.3	± .9
Multiplication	3.5	1.3	± 1.95	± .8
Division, all V_1	−16	−6.2	± 15	± 5.8
Division, $V_1 > 45$	3.4	1.3	± 1.65	± .6

Table 7.2: Summary of Arithmetic Functions, Processing Speed=0.8MIPS, LSB is for 8 bits of resolution

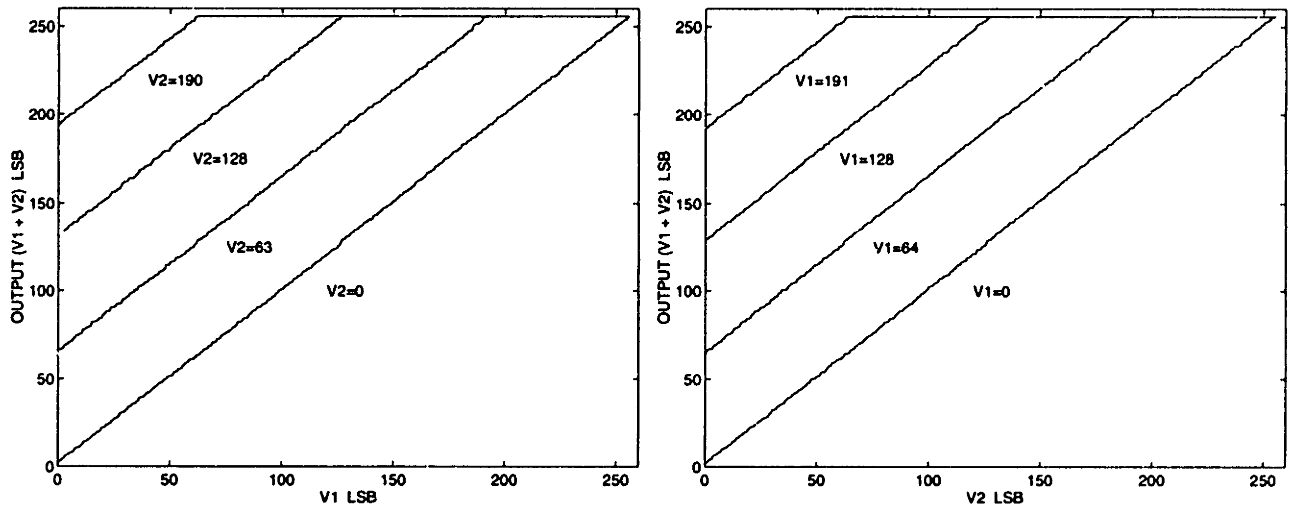


Figure 7.12: Addition Data, Ramp Inputs, F=4.0MHz, LSB is for 8 bits of resolution

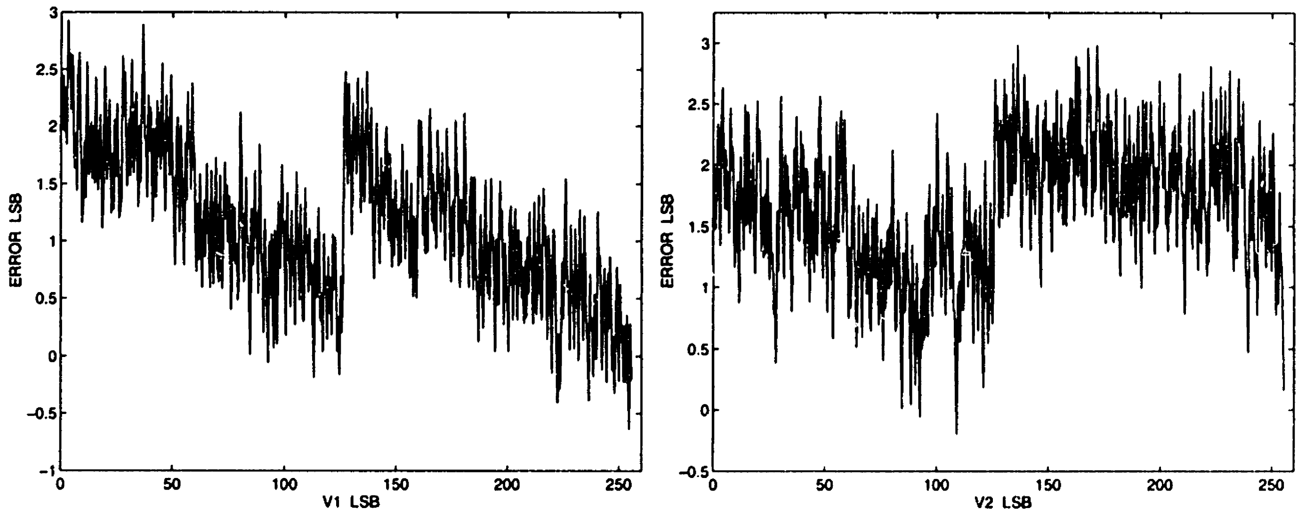


Figure 7.13: Addition Data, Actual-Ideal, F=4.0MH, LSB is for 8 bits of resolution

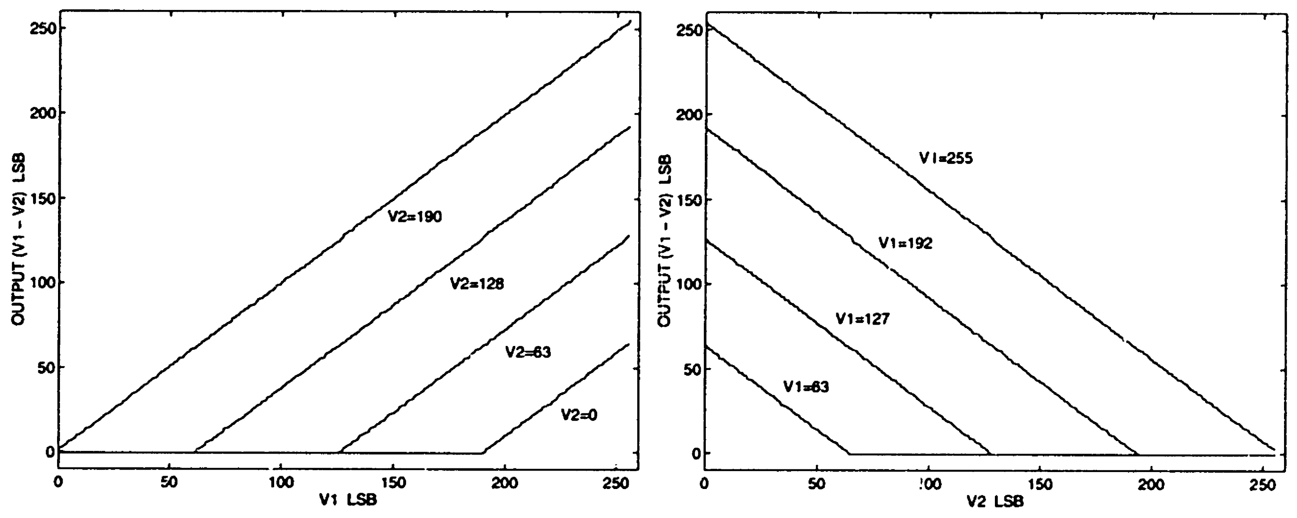


Figure 7.14: Subtraction Data, Ramp Inputs, F=4.0MHz, LSB is for 8 bits of resolution

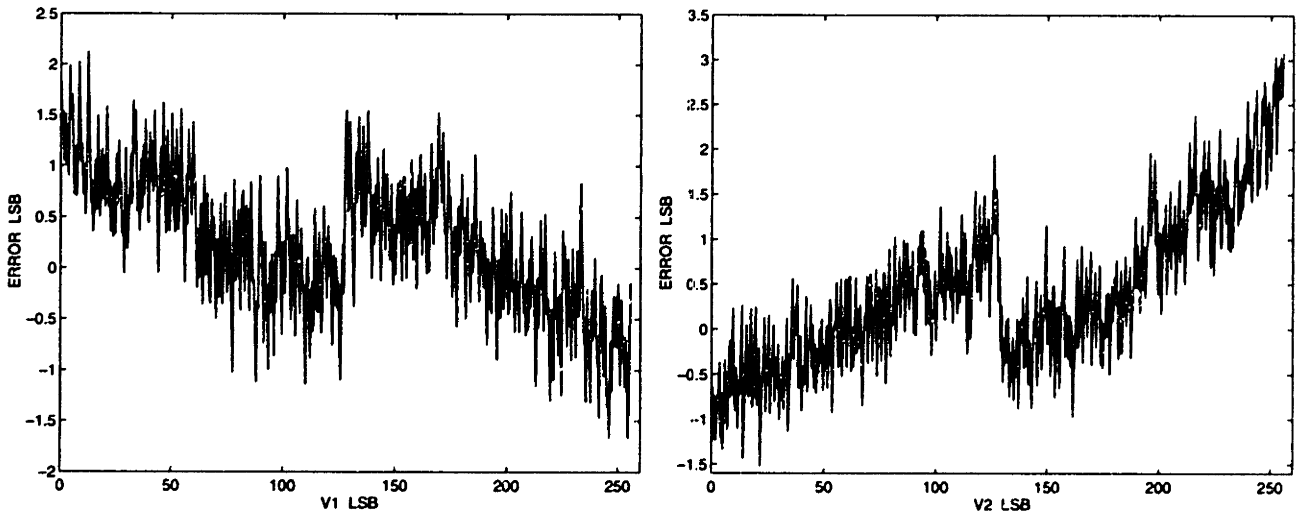


Figure 7.15: Subtraction Data, Actual-Ideal, F=4.0MHz, LSB is for 8 bits of resolution

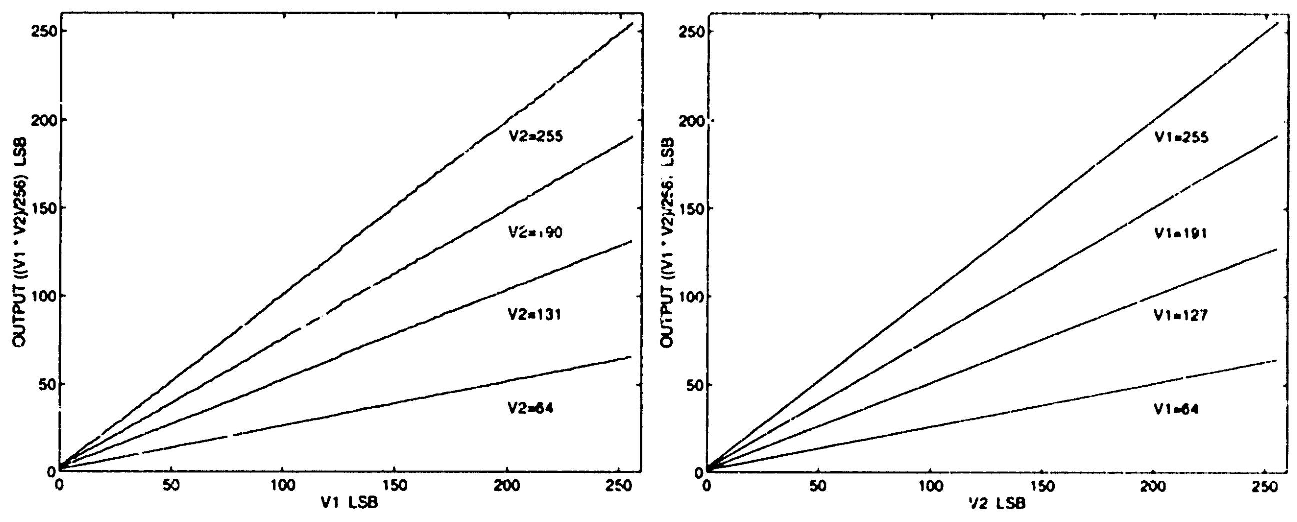


Figure 7.16: Multiplication Data, Ramp Inputs, F=4.0MHz, LSB is for 8 bits of resolution

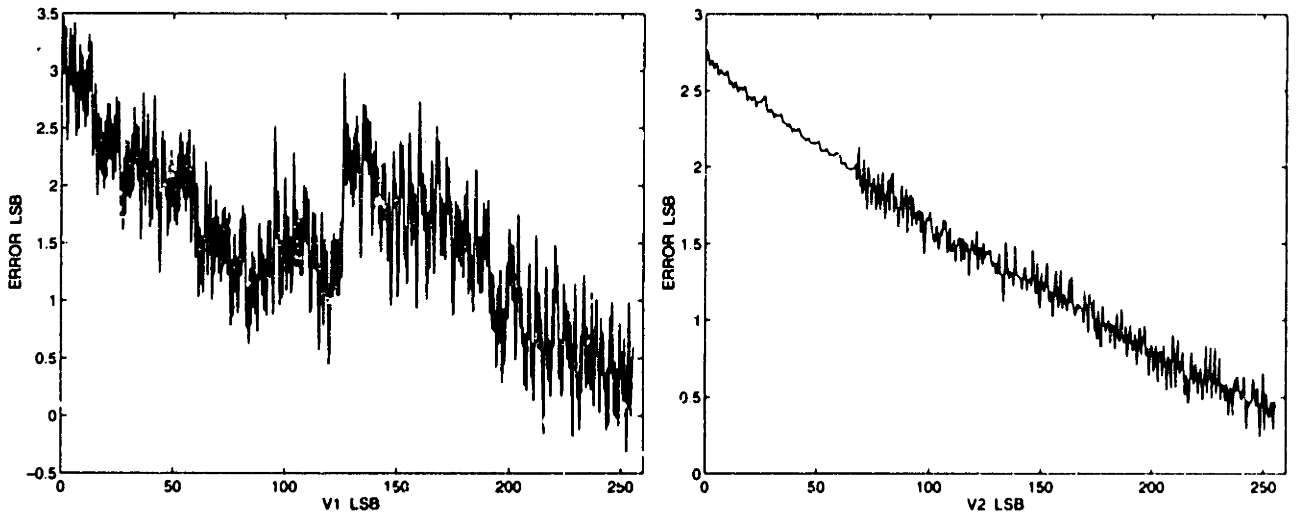


Figure 7.17: Multiplication Data, Actual-Ideal, F=4.0MHz, LSB is for 8 bits of resolution

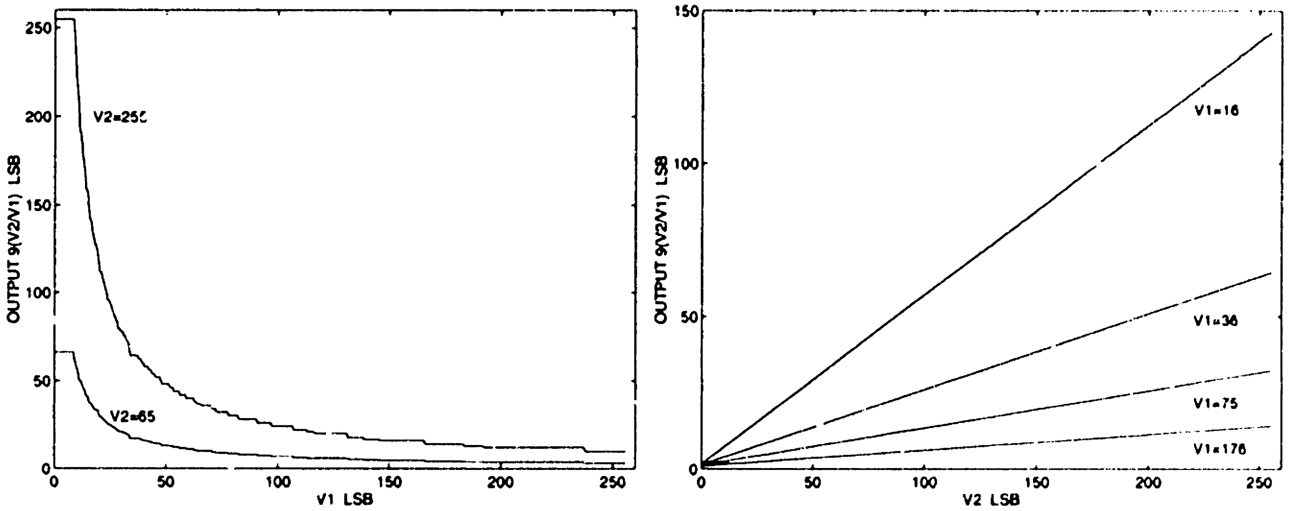


Figure 7.18: Division Data, Ramp Inputs, F=4.0MHz, LSB is for 8 bits of resolution

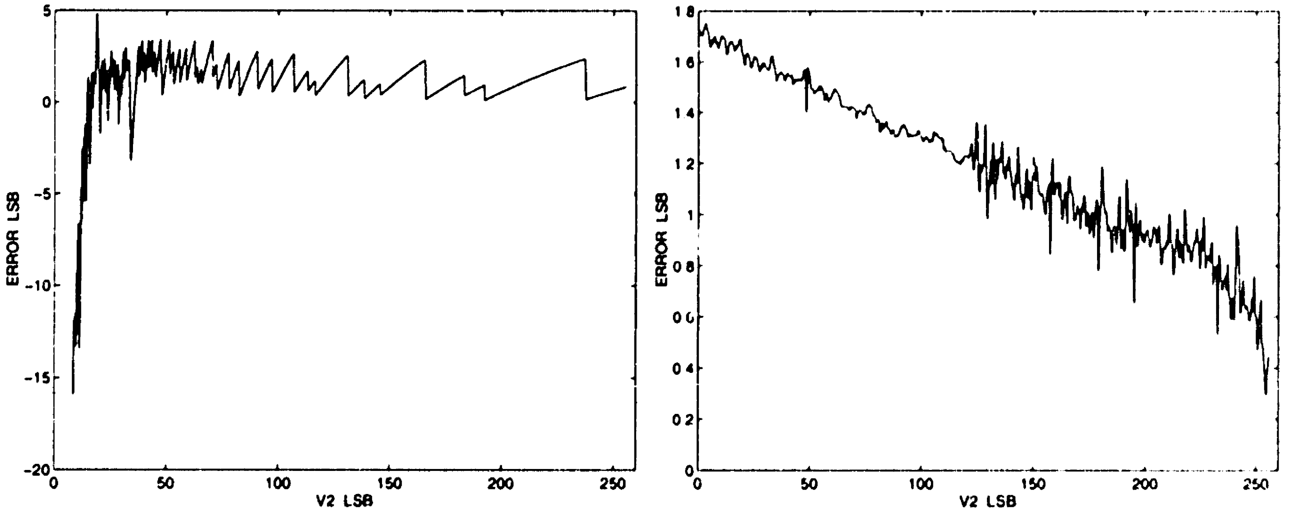


Figure 7.19: Division Data, Actual-Ideal, F=4.0MHz, LSB is for 8 bits of resolution

7.5 Edge Detection Algorithm

To test the ADAP array chip, an edge detection algorithm was performed. The resulting edge map was then compared to an edge map generated by running the same algorithm on a computer. The edge detection algorithm is:

$$Edge(x,y) = \frac{K_2 Pixel(x+1,y) + K_2 Pixel(x+2,y)}{K_1 Pixel(x-1,y) + K_1 Pixel(x-2,y)} \tag{7.1}$$

This algorithm will find positive edges, in which pixel values increase from left to right. Negative edges can be found by using the multiplicative inverse of the equation. Often, the pixel values in rows $y-1$ and $y+1$ are also used to calculate $Edge(x,y)$. However, the 5x5 array chip was too small to perform the extra calculations. The values of K_1 and K_2 are chosen according to the light levels present in the image. For the image used in this test, K_1=31, and $K_2 = 154$. V_{G+K} was set to 1.507V, giving a K of 9. These constants are not symmetric around the pixel; they are used to scale the pixel values so that the division operation will take place in the most nonlinear region of operation, as described in Section 2.2. The array was clocked at 4MHz, the same speed at which the arithmetic operations were tested. The computer program and array programming are described in Appendix C.

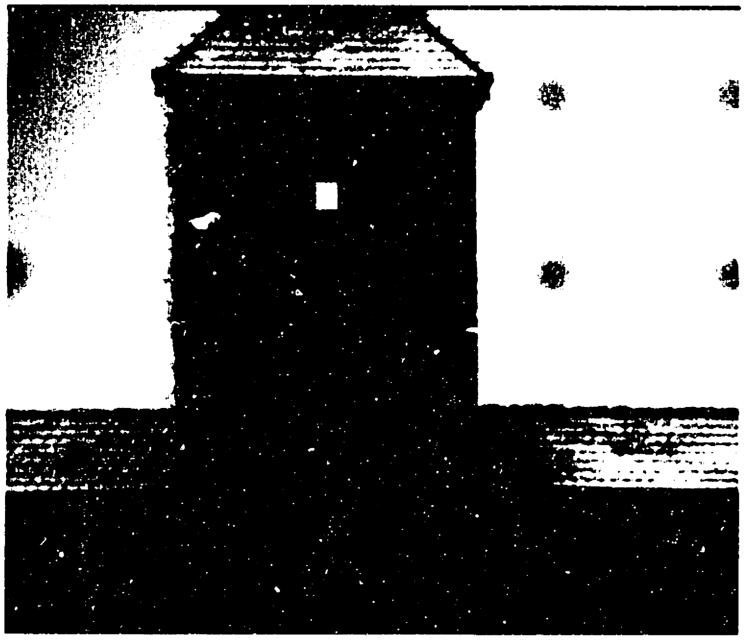


Figure 7.20: Input Image for Edge Map Algorithm, Courtesy of CMU's Calibrated Imaging Laboratory

The image used for the test is shown in Figure 7.20. The edge map (of positive edges in this case) generated by running the algorithm in software is shown in Figure 7.22. The image was sent into the ADAP array chip using a Data Precision 8200 as described in Section 6.3.4. The resulting edge map is shown in Figure 7.23. There are two properties of the edges in the two edge maps that can be compared: their location within the edge map and the actual edge value. To determine edge locations, it is assumed that an edge exists wherever the edge value exceeds a certain threshold. Assuming a threshold of 128 (half of the 0-255 range of edge values), both the software edge map and the ADAP edge map have edges in the same locations. The actual edge values at these locations were not exactly the same. On average, the edge values in the software generated edge map were about 5% higher than those in the ADAP edge map. This is probably due to the fact that the edge value is the result of the division operation in Equation 7.1. One limitation of ADAP's division operation is that its output can never be higher than the numerator. As a result, there is a set of input values for which the correct edge value is higher than the numerator in Equation 7.1.

The edge detection algorithm was also run with a picture of a car. The original picture is shown in Figure 7.21, the edge map produced in software is shown in Figure 7.24, and the edge map produced by ADAP is shown in Figure 7.25. These edge maps are sharper than those of the previous image because the image was sharper, with better defined black and white regions. One limitation of ADAP can be seen in its edge map. Because of offsets in the ADAP ALU and electrical noise, the edge values which should be zero are actually often non-zero, ranging from zero to ten. This shows up in the edge map in Figure 7.25 as light grey.

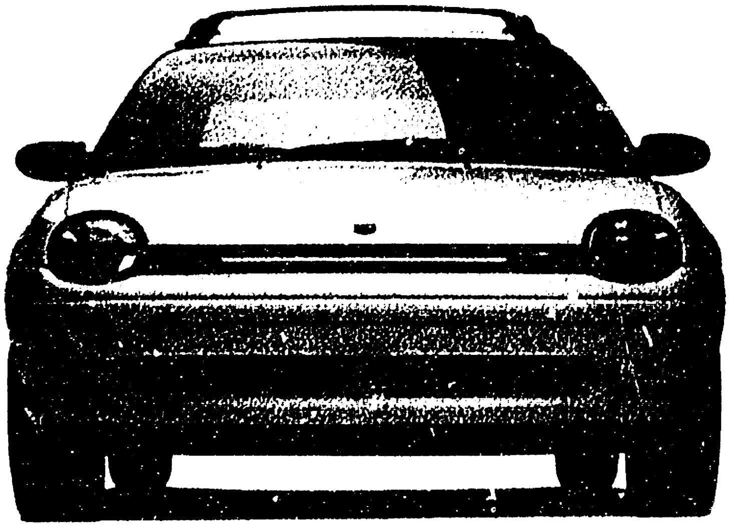


Figure 7.21: Car Image for Edge Map Algorithm, Courtesy of Dodge

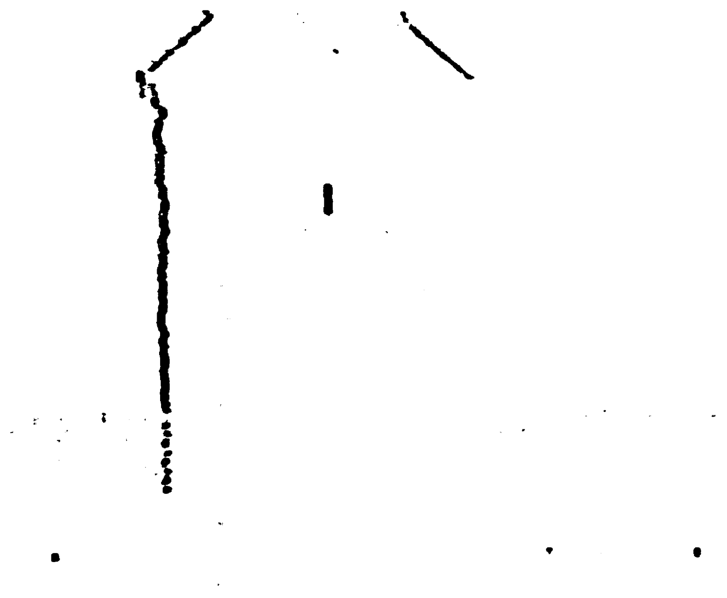


Figure 7.22: Edge Map Generated in Software

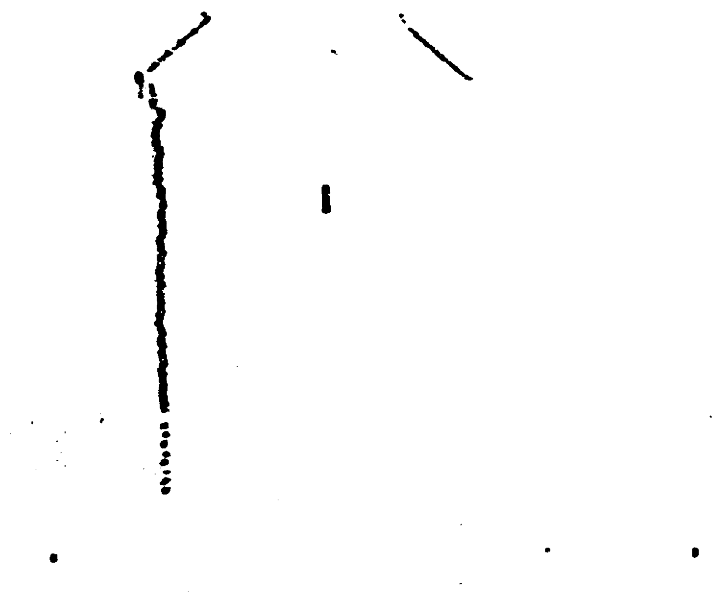


Figure 7.23: Edge Map Generated by ADAP Array Chip

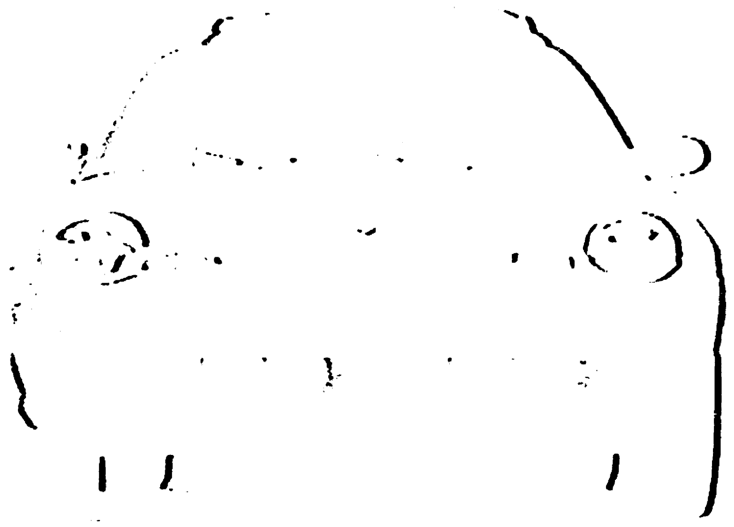


Figure 7.24: Edge Map Generated in Software

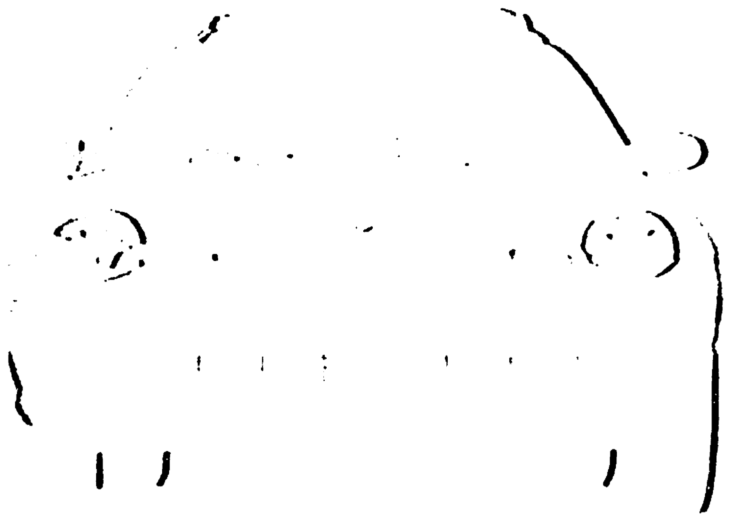


Figure 7.25: Edge Map Generated by ADAP Array Chip

7.6 Sub-Pixel Resolution Algorithm

The sub-pixel resolution algorithm was also tested on the ADAP array. The algorithm determines the edge location with sub-pixel resolution using an edge value and the two neighboring edge values, as described in Section 2.5. The equation for this algorithm, Equation 2.11, is:

$$X_{Actual} = \frac{P_2 - .75P_1 - .25P_3}{P_2 - .5P_1 - .5P_3} \tag{7.2}$$

The edge maps for the image in Figure 7.20 were used as the input for this algorithm because the physical location of several objects in the scene had been measured with great accuracy when the image was created. These are the points labeled "13" and "14" in Figure 7.26. For point 13, the right and left edge locations should be 62.22 and 315.22; the sub-pixel resolution algorithm gave edge locations of 62.55 and 315.22. For point 14, the right and left edge locations should be 167.47 and 419.60; the sub-pixel resolution algorithm gave edge locations of 167.44 and 419.55. Thus, for three of the points, the algorithm increased the resolution by a factor of about 9 (to the closest .11). For the fourth point, the resolution was increased by a factor of about 3 (the error went to .33 from 1). Other points in the picture were examined, and the average improvement in resolution was a factor of 4 to 5 (closest .22).

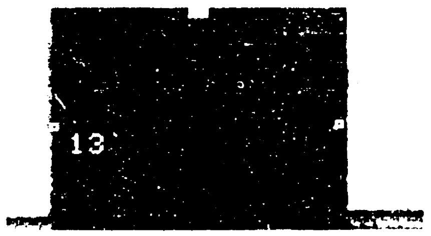


Figure 7.26: Location of Points for Sub-Pixel Algorithm

7.7 Power

The power used by ADAP depends upon the clock frequency. At 1MHz, a processor cell uses 1.6mW. At 5MHz, the processor cell uses 1.9mW. Assuming a linear dependence of power upon frequency, $P = 0.075F + 1.525$, where P is the power in mW, and F is the frequency in MHz.

The frequency-dependent power is used in the digital circuitry. In the analog portion of the circuitry, the power is distributed as shown in Table 7.3.

Circuit	Power
All Analog Circuits	$1500\mu W$
A/D	$395\mu W$
D/A	$200\mu W$
ALU S&H's	$645\mu W$
Analog Storage Unit S&H	$260\mu W$

Table 7.3: ADAP Analog Power Use

Chapter 8

Conclusions

This chapter summarizes ADAP's performance, compares it to comparable digital processors at both the chip and system levels, and suggests future improvements. It ends with a summary of the primary results.

8.1 Performance Summary

This thesis has demonstrated an analog array processor architecture. The processor was fabricated in the HP 0.8μm triple metal CMOS process. The processor has a data storage unit which can store data to 8 bits of accuracy, a switch fabric, and an ALU which can perform addition, subtraction, multiplication, and division to 7 bits of accuracy at a speed of 0.8 MIPS. The processor consumes 1.825mW of power at this speed and uses 270μm by 700μm of silicon area. The number of processors which can be combined on a single chip is limited only by the chip area. An array of these processors was used to successfully perform an edge detection algorithm, and a sub-pixel resolution algorithm executed on the array was able to increase the resolution of the edge locations by a factor of four.

8.2 ADAP-IMAP-HDPP Comparison: Processor Level

The digital processors chosen for comparison with ADAP are IMAP and HDPP. IMAP, described in [28], is a SIMD processor intended for vision processing applications; it is designed in a .55μm double metal layer BiCMOS process. IMAP consists of a linear array of 64 processor cells, each of which also has 8kbytes of SRAM memory. The array has one instruction port, one input port, and one output port. The entire array consumes 14.2mm by 13.6mm of silicon area and consumes 3.4W of power. The processors are clocked at 40MHz and require 11 cycles to perform a multiplication operation. Thus, each processor cell has an area of 3mm^2, consumes 53mW of power, and can process data at 3.63 MIPS. All arithmetic operations are performed with an 8-bit ALU.

Another digital processor designed for vision processing, HDPP, is presented in [41]. It has a similar SIMD architecture to IMAP, but employs 128 bits of DRAM memory instead of the SRAM used in IMAP and uses a 64 by 64 rectangular array of bit-oriented processors rather than 64 byte-oriented processors. HDPP can store an entire 64x64 image (or a 64x64 *part* of an image) in the array, storing one pixel in the memory of each processor. HDPP has not yet been fabricated, so there are no measured performance numbers. However, simulations predict a processor area of .015mm^2 (including wiring), power consumption of .06mW, and a processing speed of .052 MIPS for 8 bit multiplication (192 cycles at 10MHz). It has been designed in a .75μm triple metal CMOS process.

Both digital processors latch in a new instruction for every instruction cycle using a special instruction data-bus for this purpose. This is different from ADAP, which loads in a *set* of instructions, each of which is stored in a different processor, when it is initialized and then processes this set of instructions repeatedly.

To compare IMAP, HDPP, and ADAP, the power consumption and MIPS of an individual processor cell were divided by the processor cell area to get a measure of how efficiently they use power and area. For this comparison MIPS measures how many million arithmetic instructions can be performed in a second by one processor. ADAP is a MIMD processor while HDPP and IMAP are SIMD processors. Since the algorithms which they implement are vision algorithms with a high degree of parallelism, they are expected to have similar array efficiencies, as explained in Section 3.3.3. The results are shown in Table 8.1.

Metric	ADAP	IMAP	HDPP
MIPS/cell	0.8	3.63	.052
Power/cell	1.825mW	53mW	0.06mW
Cell Area	0.19 mm^2	3mm^2	0.015mm^2
MIPS/mm^2	4.2	1.2	3.5
MIPS/mW	0.43	0.068	0.86

Table 8.1: Comparison of IMAP, HDPP, and ADAP Performance per Cell

ADAP is more than 6 times more efficient than IMAP in its use of power, and ADAP gets more than 3 times more performance from a given area of silicon. ADAP's advantage comes from the fact that it's ALU uses area and power more efficiently than IMAP; it uses the same circuit to perform all four arithmetic functions. ADAP and HDPP are more closely matched. ADAP gets about 20% more performance from the silicon area as HDPP, but HDPP uses half as much power for its processing.

IMAP and HDPP do have some advantages. They have far more memory available for each processor and can perform binary operations, such as AND, OR, and bit shifting, on the data. One limitation shared by all processors is the inability to perform IF/THEN instructions.

Although IMAP and HDPP are clocked at a higher speed than ADAP, they use more cycles (IMAP uses 11 for multiplication) to perform arithmetic operations. IMAP's and HDPP's designers did not specify how many cycles a division operation would require; it is safe to assume that such an operation would require at least as many cycles as a multiplication operation. IMAP, HDPP, and ADAP can process data at higher speeds if they only need to perform addition and subtraction. Both digital and analog systems take a performance hit in terms of speed, area, or power as more complex functionality is added. Although the performance degradation cannot be rigorously quantified for every case, for IMAP, HDPP, and ADAP the division/multiplication functionality cuts the processing speed by about an order of magnitude.

IMAP and HDPP use higher performance processes, having a gate length of 0.55μm or 0.75μm compared to ADAP's 0.8μm process. It is hard to quantitatively scale the performance figures for this process difference, but it is safe to assume that ADAP could achieve a higher performance if it was designed in IMAP's or HDPP's process.

8.3 ADAP-IMAP Comparison: System Level

At the system level, IMAP has several advantages over ADAP. These advantages are primarily the result of the flexibility and robustness inherent to digital systems.

- Since IMAP is clocked at a higher rate, it can handle higher video rates than ADAP.
- High temperatures will degrade the performance of both IMAP and ADAP. However, a digital circuit such as IMAP will usually degrade more gracefully (primarily by slowing down) than an analog circuit such as ADAP, which will also lose accuracy as well as speed with temperature.
- Because it uses image data in a digital format, IMAP can perform several operations on the same image data and even store the image for later use. Once ADAP processes an image, the image is lost; only the results of the operation are left.
- ADAP expects its analog inputs to be within a certain voltage range. If the video outputs are not within this range, then some sort of conversion circuit (an A/A chip) must be placed between the imager and ADAP.

One goal of this thesis was to study the role of analog processing in vision systems in order to determine where in the datapath the A/D operation should take place. There are some functions which digital systems can perform very well at high speeds. These functions include bit shifting, simple logical functions, sorting data, and moving data between memory locations. Digital systems can also perform floating point arithmetic, but at a high cost in silicon area and power. Analog circuits are very bad at logical operations and moving data around, but they can perform floating point operations very quickly with small amounts of power and area *if* only 7 to 8 bits of accuracy are required. Thus, a good design technique for vision systems is to perform any low precision, floating point operations with analog circuits on the image data before performing the A/D conversion. The remaining functions wil)e those that are best suited for a digital processor.

8.4 Future Work

There are a number of improvements which can be made to the ADAP processor.

- The improved clock delay circuit mentioned in Section 7.1.2 should be implemented in order to improve ADAP's speed.
- Currently, the ability to shift or hold data in the cell analog storage units can only be controlled on a row by row basis. Some algorithms would benefit from the ability to control this function on a cell by cell basis. A way to do this is described in Appendix B.3.
- Although it is easier to generate and collect output from ADAP in digital form, ADAP also generates analog outputs so that several chips can be interconnected. These have a problem driving the pad and PCB trace capacitance fast enough to transfer data at 1 MIPS. One solution which increases the data rate is to modify the timing as suggested in Appendix B.1 so that 9 clock phases rather than 2 are used for the sample and hold output to settle to its final value.
- The question of processor utilization in MIMD and SIMD array processors deserves further investigation. One way to do this is to implement several vision algorithms on both ADAP and HDPP and compare their processor utilizations.
- To use ADAP or any image processor in an actual system, it is necessary to convert the imager output format to the processor's input format. For research-oriented systems, this converter can be a complicated system in itself [58]. For a commercial system employing ADAP, an NTSC→ADAP chip or chipset would be necessary.
- One problem with any special image processor, analog or digital, is that most commercial imagers send out data a row at a time. However, most robust edge detection algorithms require pixel values from several different rows to calculate an edge value for one pixel. This means that pixel values must be temporarily stored in a shift register. There are several ways to solve this problem. One approach is to build vision processors with special shift registers. This has the disadvantage of requiring the shift register size, and thus the maximum image size, to be determined ahead of time. The solution employed by the HDPP processor is to store the entire image in the processor, using one processor per pixel. Although this does solve the problem of accessing a pixel's nearest neighbors, it does so at the cost of storing the entire image in the processor. Another solution is to build special imagers with parallel outputs. A third solution would be a non-destructive addressable imager, one in which certain columns or rows could be read out several times before the pixel values changed. This would be similar to a ROM in which the memory locations would be written, only at user-determined times, by the light hitting the chip. Finally, a special interface chip could be developed that would have a very flexible set of shift registers that would hold pixel values and present them to the processor in the proper order.

Finally, it would also be helpful to add the ability to perform an IF/THEN instruction. However, it could be very costly in terms of area, power, speed, and complexity. ADAP's current design essentially takes an analog circuit and makes its inputs, outputs, and references programmable. Adding the ability to perform IF instructions would add an entire new level of functionality and complexity, which would have a significant cost in area, power, or speed. A better tradeoff might be to simply enhance ADAP's current performance and let the digital system controlling ADAP perform the logical operations. An obvious way to explore this issue is to build a complete stereo vision system incorporating ADAP to test it under real-world conditions.

8.5 Thesis Summary

This thesis has demonstrated:

- An analog array processor, ADAP, can perform several early vision tasks such as edge detection and sub-pixel edge resolution determination. It does this with a new type of analog ALU accurate to between 7 and 8 bits which combines an A/D and a D/A to perform addition, subtraction, multiplication, and division.
- ADAP performs these functions with greater power and area efficiency than comparable digital image processors.
- The performance of both analog and digital systems can decline by an order of magnitude when a new level of functionality is added.
- The proper role of analog processing in vision systems is to perform the low resolution floating point arithmetic on the image data before performing the A/D operation.
- A key area for further study in the design of vision systems is the interface between the imager and the processor.

Appendix A

Error Analysis

This appendix analyzes the A/D error sources. The analysis can be used, with a few changes, to the D/A and sample and hold circuits as well.

A.1 A/D

During each cycle of the A/D, several sources produce errors. These errors can be shown by plotting the output of the A/D circuit versus its input for one cycle. Figure A.1 shows this plot for a circuit with no errors. The A/D circuit subtracts the offset V_G from its input, multiplies that by 2, subtracts either nothing or $2V_{REF}$, and re-adds the offset V_G. It subtracts nothing for inputs below V_{MID} and $2V_{REF}$ for inputs above V_{MID}. V_{MID} is equal to $V_G + V_{REF}$. To see how this graph can be altered by errors, the operation of the A/D will be examined.

In Figure A.2, when C_1 and C_2 are sampling V_{IN} while ϕ_1 is high, the charge stored on the top plates of the capacitors is:

$$Q_{C1+C2} = (V_{IN} - V_-)(C_1 + C_2) \tag{A.1}$$

$$V_- = V_+ + V_{offset} \tag{A.2}$$

$$Q_{C1+C2} = (V_{IN} - (V_+ + V_{offset}))(C_1 + C_2) \tag{A.3}$$

The voltage V_{offset} is defined to be the opamp offset when it is in unity gain configuration and $V_+ = 0.945$V. When ϕ_1 goes low, the switch (an NMOS transistor) that connects the output of the opamp to its negative input injects a charge $-\Delta Q$ into the bottom plates of the two capacitors. The magnitude of ΔQ is dependent upon V_{IN}, but this dependence is not easily quantified. From now on, ΔQ will be assumed to represent ΔQ_{MAX}. Once this switch opens, no more charge can flow to the node at the negative input of the opamp. Therefore, the only error due to switch charge injection is from the opening of the feedback switch. The charge on the capacitors is now:

$$Q_{C1+C2} = (V_{IN} - (V_+ + V_{offset}))(C_1 + C_2) + \Delta Q \tag{A.4}$$

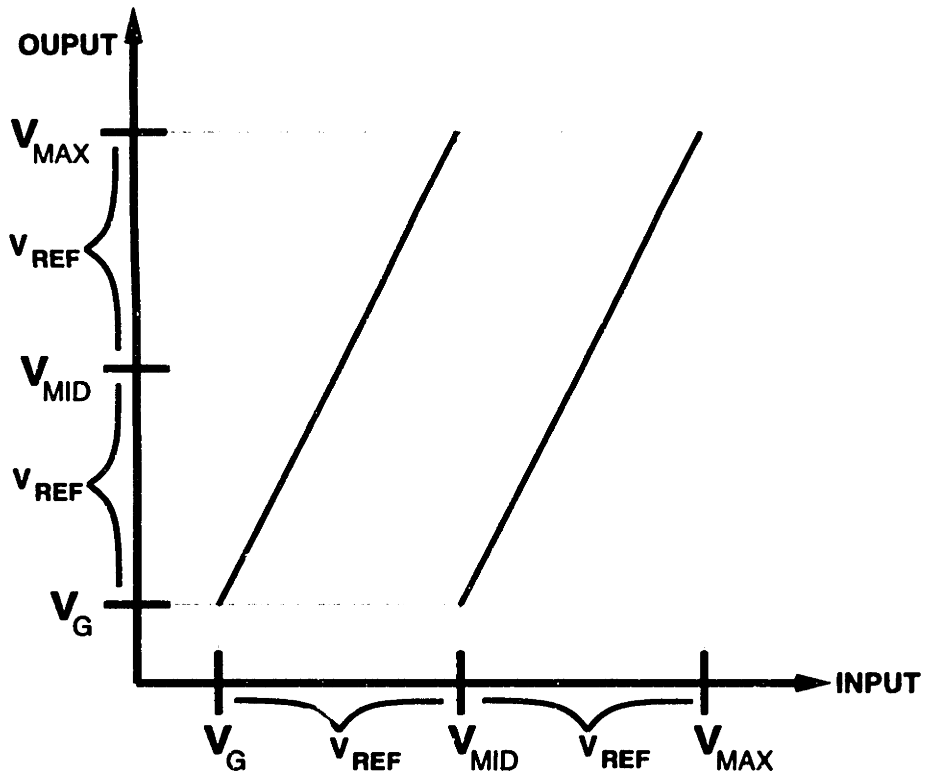


Figure A.1: A/D Residue Plot: No Errors

While both ϕ_1 and ϕ_2 are low, the top plates of C_1 and C_2 are tied to $V_G + VREF$. As a result, the voltage at the negative input is:

$$\begin{aligned} V_- &= (V_G + V_{REF}) - \Delta V_{C1,C2} && \text{(A.5)} \\ \Delta V_{C1,C2} &= Q_{C1+C2}/(C_1 + C_2) && \text{(A.6)} \\ V_- &= (V_G + V_{REF}) - ((V_{IN} - (V_+ + V_{offset}))(C_1 + C_2) + \Delta Q)/(C_1 + C_2) && \text{(A.7)} \\ V_- &= V_{REF} - V_{IN} + V_G + V_+ + V_{offset} - \Delta Q/(C_1 + C_2) && \text{(A.8)} \end{aligned}$$

Equation A.8 gives us the new voltage at V_-. If this is below the opamp trip point, the opamp's output will slew up. If it is above the trip point, the opamp's output will slew down. The opamp's trip point at V_- is:

$$\begin{aligned} V_{trip\ point} &= V_+ + V_{offset} + (-V_E) && \text{(A.9)} \\ & && \text{(A.10)} \end{aligned}$$

The voltage $-V_E$ is the voltage difference needed at the negative input to move the opamp output from its unity gain feedback output ($\approx V_+$) to its halfway point ($\approx V_{MID}$). It is expected to be about 0.5mV.

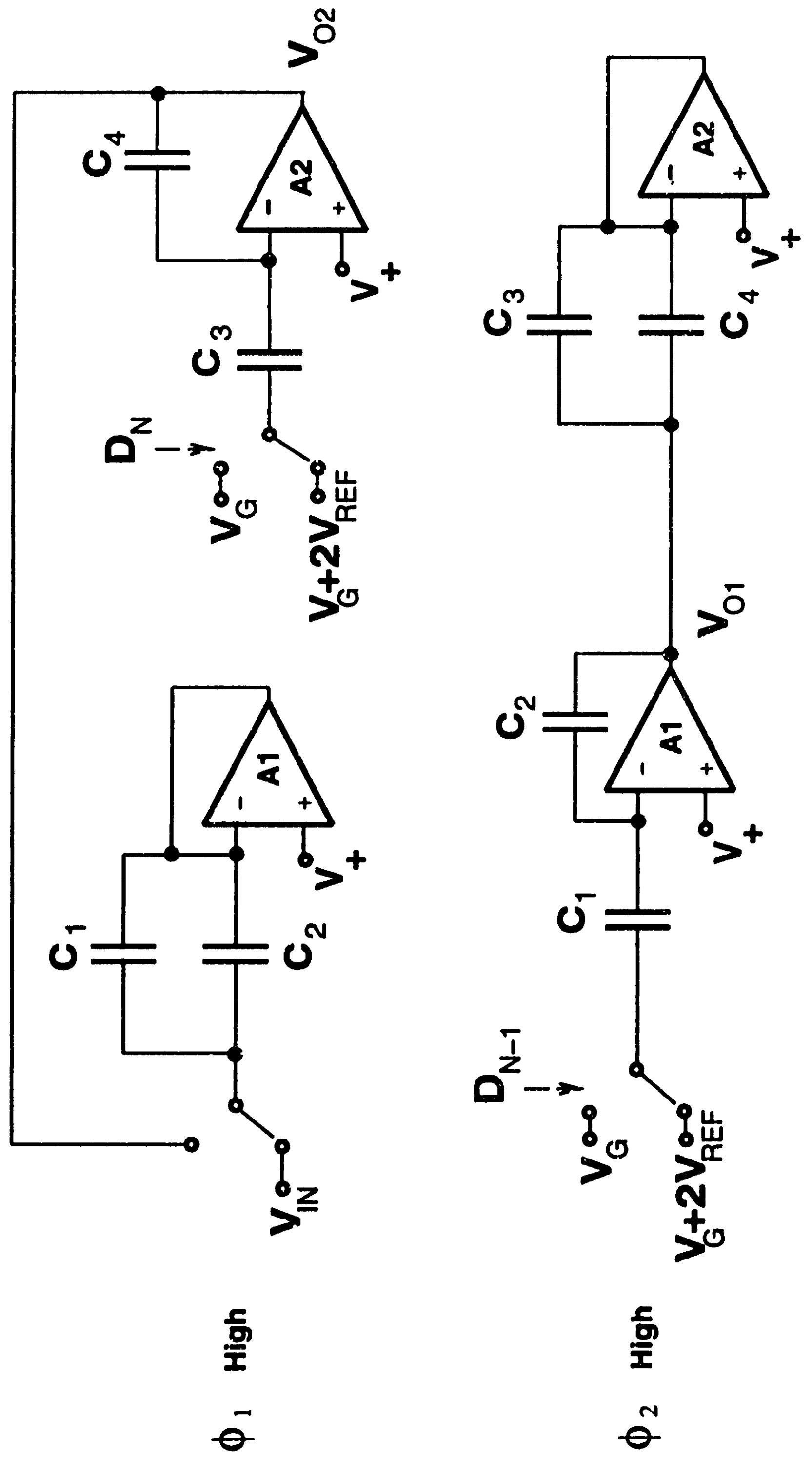


Figure A.2: A/D Schematic

The trip point voltage is then subtracted from Equation A.8:

$$\begin{aligned}
V_{diff} &= V_{-,New} - V_{trip\ point} && \text{(A.11)}\\
V_{diff} &= (V_{REF} + V_G + V_+ - V_{IN} + V_{offset} - \Delta Q/(C_1 + C_2)) - (V_+ + V_{offset} - V_E) && \\
V_{diff} &= (V_{REF} + V_G) - V_{IN} - \Delta Q/(C_1 + C_2) + V_E && \text{(A.12)}\\
V_{diff} &= (V_{MID} - V_{IN}) - \Delta Q/(C_1 + C_2) + V_E && \text{(A.13)}
\end{aligned}$$

Thus, there is an error of $-\Delta Q/(C_1 + C_2) + V_E$ in detecting the midpoint of the input range.

When ϕ_2 goes high, for input voltages below V_{MID}, the top plate of C_1 is connected to V_G. The charge on the top plate of C_2 is then:

$$\begin{aligned}
Q_{C2} &= Q_{C1+C2} - Q_{C1} && \text{(A.14)}\\
Q_{C1} &= (V_G - V_-)C_1 && \text{(A.15)}\\
V_- &= V_+ + V_{offset} - V_e && \text{(A.16)}\\
Q_{C1} &= (V_G - (V_+ + V_{offset} - V_e))C_1 && \text{(A.17)}\\
Q_{C1} &= V_G C_1 - V_+ C_1 - V_{offset} C_1 + V_e C_1 && \text{(A.18)}\\
Q_{C2} &= (V_{IN} - (V_+ + V_{offset}))(C_1 + C_2) + \Delta Q - V_G C_1 + V_+ C_1 + V_{offset} C_1 - V_e C_1 &&
\end{aligned}$$

The variable V_e is used instead of V_E since it is now not a constant, but rather function of V_{OUT}. Its dependence on V_{OUT} will be taken into account later.

There is an additional charge error that results from the parasitic capacitance C_P at V_-. This includes both the gate capacitance of the opamp's input transistor as well as any parasitic capacitance between metal lines. Since the voltage at this node changes by V_e, the charge on it changes by $V_e C_P$. Subtracting this from Q_{C2} yields:

$$\begin{aligned}
Q_{C2} &= (V_{IN} - (V_+ + V_{offset}))(C_1 + C_2) + \Delta Q - V_G C_1 + V_+ C_1 + V_{offset} C_1 - V_e C_1 - V_e C_P\\
Q_{C2} &= (V_{IN} - (V_+ + V_{offset}))(C_1 + C_2) + \Delta Q - V_G C_1 + V_+ C_1 + V_{offset} C_1 - V_e(C_1 + C_P)
\end{aligned}$$

To get the voltage across C_2, Q_{C2} is divided by C_2. After simplifying terms, this yields:

$$V_{across\ C2} = \frac{C_1 + C_2}{C_2}(V_{IN} - V_G) + \Delta Q/C_2 - V_{offset} + V_G - V_+ - \frac{C_1 + C_P}{C_2}V_e \qquad \text{(A.19)}$$

This is added to the voltage at V_-, Equation A.17, to get the voltage at the output of the opamp:

$$\begin{aligned}
V_{out} &= \frac{C_1 + C_2}{C_2}(V_{IN} - V_G) + \Delta Q/C_2 - V_{offset} + V_G - V_+ - \frac{C_1 + C_P}{C_2}V_e + (V_+ + V_{offset} - V_e) &&\\
V_{out} &= \frac{C_1 + C_2}{C_2}(V_{IN} - V_G) + V_G + \Delta Q/C_2 - \frac{C_1 + C_P}{C_2}V_e - V_e && \text{(A.20)}
\end{aligned}$$

Letting $C = C_2$, and $C_1/C_2 = 1 + \epsilon$, where ϵ is the capacitor mismatch:

$$V_{out} = (2+\epsilon)(V_{IN} - V_G) + V_G + \Delta Q/C - (2 + \epsilon + C_P/C_2)V_e \qquad (A.21)$$

However, V_e is not a constant; it is actually a function of V_{OUT}:

$$V_e \approx (1/A_v)(V_{OUT} - V_+) \qquad (A.22)$$

Using $f = (2 + \epsilon + C_P/C_2)(1/A_V)$:

$$\begin{aligned} V_{out} &= (2+\epsilon)(V_{IN} - V_G) + V_G + \Delta Q/C - f(V_{out} - V_G) && (A.23) \\ V_{out} &= \frac{2+\epsilon}{1+f}(V_{IN} - V_G) + V_G + \frac{1}{1+f}\Delta Q/C && (A.24) \end{aligned}$$

For the case of input voltages above the midpoint, the output is:

$$V_{out} = \frac{2+\epsilon}{1+f}(V_{IN} - V_G) - \frac{1+\epsilon}{1+f}2V_{REF} + V_G + \frac{1}{1+f}\Delta Q/C \qquad (A.25)$$

This is shown in Figure A.3[1]. With the current circuit configuration, f is expected to be about 10^{-3}.

The vertical distance between the endpoints of the plot at $V_{MID} - (-\Delta Q/2C + V_E)$ should be $2V_{REF}$. Any deviation from this value contributes towards DNL. In this case, they are off by a factor of $\frac{(\epsilon-2f)V_{REF}}{1+f}$. Thus, the main factors leading to DNL in the A/D would appear to be capacitor mismatch and the finite gain of the opamp, not charge injection. However, the charge injection is actually signal-dependent although it is modeled here as signal-independent. So, in reality, it would contribute to the DNL as well. Charge injection also contributes to the A/D's INL, of course.

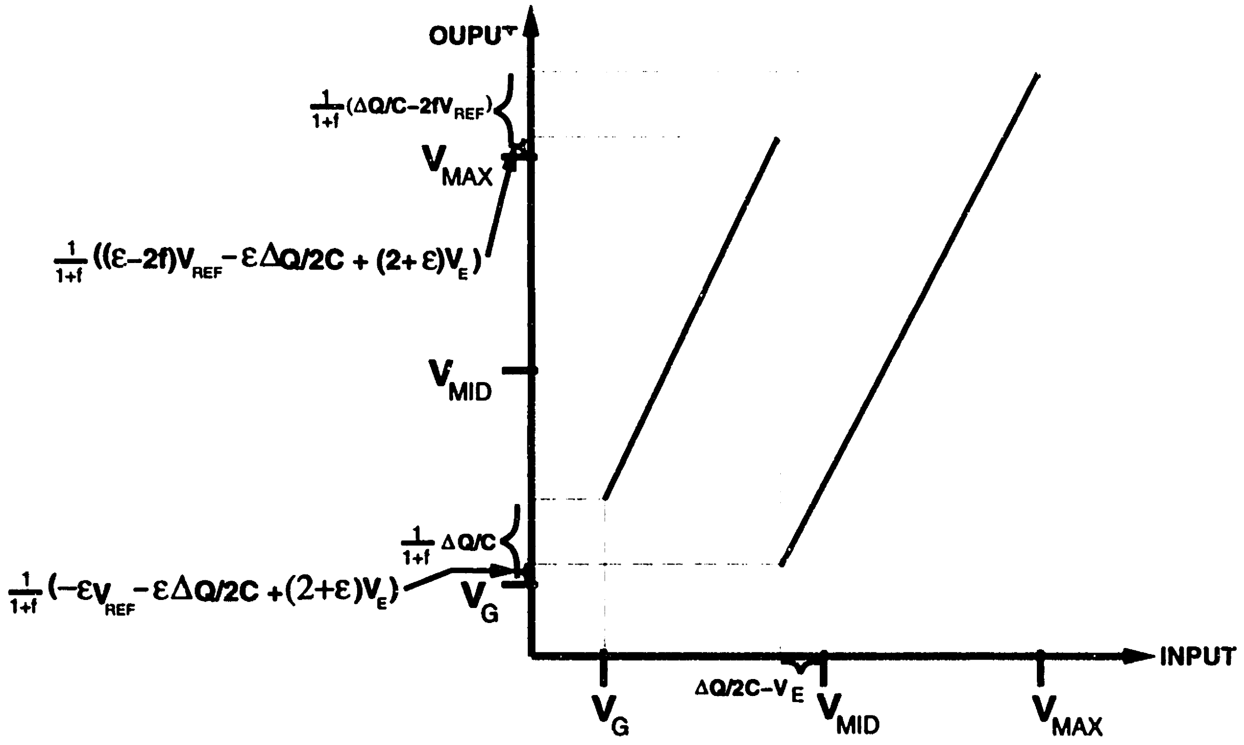
The offset of the opamp cancels out and does not affect the accuracy of the A/D operation.

The charge injection from the feedback switch was minimized by using a minimum size device (1.2um/.8um) and by using an ON voltage of about 2.5V instead of 5V. This reduced the amount of charge stored in the transistor's parasitic capacitances and channel.

This analysis also applies to the D/A. Its errors also arise from capacitor mismatch, charge injection from the feedback switch, parasitic capacitance, and the finite gain of the opamp.

The sample and hold circuits have similar sources of errors, except that there is no capacitor mismatch since there is only one capacitor. The charge injection shows up as a positive offset.

[1]All errors are shown to be positive. However, depending upon the magnitude and sign of ϵ, V_E, f, and ΔQ, some of them might be negative.



OUPUT
$\frac{1}{1+f}(\Delta Q/C-2fV_{REF})$
V_{MAX}
$\frac{1}{1+f}((\varepsilon-2f)V_{REF}-\varepsilon\Delta Q/2C+(2+\varepsilon)V_E)$
V_{MID}
$\frac{1}{1+f}\Delta Q/C$
$\frac{1}{1+f}(-\varepsilon V_{REF}-\varepsilon\Delta Q/2C+(2+\varepsilon)V_E)$
V_G
INPUT
V_G
$\Delta Q/2C-V_E$
V_{MID}
V_{MAX}

Figure A.3: A/D Residue Plot: Charge Injection, Capacitor Mismatch, and Opamp Gain Errors

A.2 ALU Quantization Error

Since the output of the ALU's D/A can only have 2^N discrete output values, the ALU has only 2^N possible output values. This will limit the performance of some algorithms. The same sort of restriction exists in a digital integer arithmetic circuit.

To quantify the quantization error. The output of the A/D, D_{MID}, is:

$$D_{MID} = \lfloor \frac{V_{IN}}{V_{REF,A/D}} 2^N \rfloor \tag{A.26}$$

$$D_{MID} = \frac{V_{IN}}{V_{REF,A/D}} 2^N - \varepsilon \tag{A.27}$$

where $\lfloor\ \rfloor$ indicates floor (nearest lower integer). ε is the difference between the real number $\frac{V_{IN}}{V_{REF,A/D}} 2^N$ and the integral output of the A/D $\lfloor \frac{V_{IN}}{V_{REF,A/D}} 2^N \rfloor$.Then, using this as the input to the D/A, the output of the D/A is:

$$V_{OUT} = \frac{D_{MID}}{2^N} V_{REF,D/A} \tag{A.28}$$

$$V_{OUT} = (\frac{V_{IN}}{V_{REF,A/D}} 2^N - \varepsilon) \frac{1}{2^N} V_{REF,D/A} \tag{A.29}$$

$$V_{OUT} = \frac{V_{IN} V_{REF,D/A}}{V_{REF,A/D}} - \varepsilon \frac{V_{REF,D/A}}{2^N} \tag{A.30}$$

$$\tag{A.31}$$

So, the quantization error Δ is $\varepsilon \frac{V_{REF,D/A}}{2^N}$. Since the maximum value of ε is one LSB of the A/D:

$$\Delta_{MAX} = \varepsilon_{MAX} \frac{V_{REF,D/A}}{2^N} \tag{A.32}$$

$$\varepsilon_{MAX} = 1 \tag{A.33}$$

$$\Delta_{MAX} = \frac{V_{REF,D/A}}{2^N} \tag{A.34}$$

So, Δ_{MAX} is simply one D/A LSB.

Appendix B

Timing Information

B.1 Cell Timing

The primary system clock for ADAP consists of two non-overlapping clock signals ϕ_1 and ϕ_2. Each clock cycle has ϕ_1 high and then ϕ_2 high for equal periods of time. Since the ADAP instruction cycle consists of 5 clock cycles, there are thus a total of 10 different clock phases. The signal V_F is high for the first two phases, and the signal V_L is n high for the last two phases. When V_{SHIFT} is asserted for an instruction cycle, it has the same timing as V_F; otherwise, it is low for the entire cycle. The clock timing is shown in Table B.1, as are the functions performed by each circuit. In the table, SHI is the input sample and hold (SH) for the analog data storage unit, and SHO is the output SH. AD1 and AD2 are the first and second opamps of the A/D. COMP shows what bit is currently the output of the comparator. BUFFI shows what bit is present at the input to the digital buffer. This value is latched at the transition between phases. BUFFO shows what bit is present at the output of the digital buffer. DA1 and DA2 are the first and second opamps of the D/A. ISH1 is the SH which holds V_2 for the A/D; ISH2 is the SH which holds V_{REF} for the A/D; and ISH3 is the SH which holds V_{REF2} for the A/D. OSH1 is the SH which holds V_2 for the D/A; OSH2 is the SH which drives the ALU's output. This is also summarized in Table B.2.

When a SH circuit is holding a value but not driving any other circuits, its function is listed as "Hold". When it is driving the input of another circuit, it is listed as "HOLD".

The one exception to the timing shown in Table B.1 occurs during a division operation. During the second sub-cycle of a division operation, AD2 samples V_{G+9} instead of the output of AD1. Meanwhile, the output of AD1, which is $V_1/2$, is sampled by ISH2.

In order for the sample and hold circuits to work correctly, it is necessary for the V_L and V_F edges to take place entirely during the non-overlapping period between ϕ_1 and ϕ_2.

One improvement that can be made for future versions of ADAP is to change the sample sub-cycle for SHO from CLK9 to CLK2. This will give it 9 sub-cycles instead of 2 sub-cycles to drive its output. This could be important when driving a signal off-chip.

Circuit	CLK1	CLK2	CLK3	CLK4	CLK5	CLK6	CLK7	CLK8	CLK9	CLK10
ϕ_1	High	Low	High	Low	High	Low	High	Low	High	Low
ϕ_2	Low	High	Low	High	Low	High	Low	High	Low	High
V_F	High	High	Low	Low	Low	Low	Low	Low	Low	Low
V_L	Low	Low	Low	Low	Low	Low	Low	Low	High	High
V_{SHIFT}	High	High	Low	Low	Low	Low	Low	Low	Low	Low
SHI	Sample	Hold	Hold	Hold	Hold	Hold	Hold	Hold	HOLD	Hold
SHO	HOLD	Hold	Hold	Hold	Hold	Hold	Hold	Hold	Sample	HOLD
ISH1	Sample	HOLD	Hold	Hold	Hold	Hold	Hold	Hold	HOLD	Hold
ISH2	X	Sample	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD
ISH3	Sample	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD
AD1	Sample Inputs	Hold V_{IN}	Sample AD2	+,2X Bit 6	Sample AD2	+,2X Bit 4	Sample AD2	+,2X Bit 2	Sample AD2	+, 2X LSB
AD2	X	Sample V_{IN}	+,2X MSB	Sample AD1	+,2X Bit 5	Sample AD1	+,2X Bit 3	Sample AD1	+,2X Bit 1	Sample AD1
COMP	X	X	MSB	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	LSB
BUFFI	X	X	MSB	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	LSB
BUFFO	LSB	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	MSB	X	X
DA1	Sample LSB	+, (1/2)X	Sample Bit 2	+, (1/2)X	Sample Bit 4	+, (1/2)X	Sample Bit 6	+, (1/2)X	X	X
DA2	X	Sample Bit 1	+, (1/2)X	Sample Bit 3	+, (1/2)X	Sample Bit 5	+, (1/2)X	Sample MSB	+,(1/2)X	X
OSH1	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD	HOLD	Sample	Hold
OSH2	HOLD	Hold	Hold	Hold	Hold	Hold	Hold	Hold	Sample	HOLD

Table B.1: ADAP Cell Timing

Circuit	Function	Clock Signals
SHI	Input Sample and Hold for the Analog Data Storage Unit	$V_F\phi_{1S}$,$V_F\phi_{1D}$, $V_F\phi_{1N}$
SHO	Output Sample and Hold for the Analog Data Storage Unit	$V_L\phi_{1S}$,$V_L\phi_{1D}$,$V_L\phi_{1N}$
ISHI1	Sample and Hold to hold V_2 for the A/D	$V_F\phi_{1S}$,$V_F\phi_{1D}$, $V_F\phi_{1N}$
ISHI2	Sample and Hold to hold V_{REF} for the A/D	$V_F\phi_{2S}$, $V_F\phi_{2D}$, $V_F\phi_{2N}$
ISHI3	Sample and Hold to hold V_{REF2} for the A/D	$V_F\phi_{1S}$,$V_F\phi_{1D}$, $V_F\phi_{1N}$
OSHI1	Sample and Hold to hold V_2 for the D/A	$V_L\phi_{1S}$,$V_L\phi_{1D}$,$V_L\phi_{1N}$
OSHI2	Sample and Hold to hold D/A's output	$V_L\phi_{1S}$,$V_L\phi_{1D}$,$V_L\phi_{1N}$
AD1	First opamp for the A/D	ϕ_{1S},ϕ_{2D}
AD2	Second opamp for the A/D	ϕ_{2S},ϕ_{1D}
COMP	Comparator for A/D	$\overline{\phi_{12}}$
BUFFI	Input at first register of digital buffer	$\overline{\phi_{12}}$
BUFFO	Output of last register of digital buffer	$\overline{\phi_{12}}$
DA1	First opamp for the D/A	ϕ_{1S},ϕ_{2D}
DA2	Second opamp for the D/A	ϕ_{2S},ϕ_{1D}

Table B.2: ADAP Subcircuit Summary

B.2 Clock Control Logic

Each of the SH circuits, the A/D, and the D/A use special clock signals that are delayed versions of the system clocks ϕ_1 and ϕ_2. These clocks are:

- $\overline{\phi_{12}}$: This is equal to $\overline{\phi_1 + \phi_2}$. It is used as the strobe for the comparator and the clock signal for the digital buffer.

- ϕ_{1S}: This is a delayed version of ϕ_1 whose high value is V_{REF} (about 2.45V). This reduced voltage range cuts down on the resulting charge injection. It is the first clock signal to change at the end of a ϕ_1 cycle. It is used as the gate voltage of the feedback switch around AD1 in the A/D and DA1 in the D/A.

- ϕ_{1D}: This is a delayed version of ϕ_1. Its delay is set so that it go low after ϕ_{1S} goes low, and so that it will go high after the comparator's output is valid. It is used for all of the other ϕ_1 switches in both the A/D and D/A.

- ϕ_{2S}: This is a delayed version of ϕ_2 whose high value is V_{REF} (about 2.45V). This reduced voltage range cuts down on the resulting charge injection. It is the first clock signal to change at the end of a ϕ_2 cycle. It is used as the gate voltage of the feedback switch around AD2 in the A/D and DA2 in the D/A.

- ϕ_{2D}: This is a delayed version of ϕ_2. Its delay is set so that it go low after ϕ_{2S} goes low, and so that it will go high after the comparator's output is valid. It is used for all of the other ϕ_2 switches in both the A/D and D/A.

- $V_F\phi_{1S}$: This is equal to V_F AND'd with ϕ_{1S}. It is used as the gate voltage of the feedback switch around the opamp in ISH1, ISH3, and SHI.
- $V_F\phi_{1D}$: This is equal to V_F AND'd with ϕ_{1D}. It is used for all of the other sample phase switches in ISH1, ISH3, and SHI.
- $V_F\phi_{1N}$: This is equal to $\overline{V_F\phi_{1D}}$. It is used for all of the hold phase switches in ISH1, ISH3, and SHI.
- $V_F\phi_{2S}$: This is equal to V_F AND'd with ϕ_{2S}. It is used as the gate voltage of the feedback switch around the opamp in ISH2.
- $V_F\phi_{2D}$: This is equal to V_F AND'd with ϕ_{2D}. It is used for all of the other sample phase switches in ISH2
- $V_F\phi_{2N}$: This is equal to $\overline{V_F\phi_{2D}}$. It is used for all of the hold phase switches in ISH2.
- $V_L\phi_{1S}$: This is equal to V_L AND'd with ϕ_{1S}. It is used as the gate voltage of the feedback switch around the opamp in OSH1, OSH2, and SHO.
- $V_L\phi_{1D}$: This is equal to V_L AND'd with ϕ_{1D}. It is used for all of the other sample phase switches in OSH1, OSH2, and SHO.
- $V_L\phi_{1N}$: This is equal to $\overline{V_L\phi_{1D}}$. It is used for all of the hold phase switches in OSH1, OSH2, and SHO.

B.3 Analog Storage Unit Timing

Quite often, the analog storage unit will be used as a shift register. However, many algorithms require a shift register that can either hold or shift. Since a cell can only perform one instruction (hold or shift), it will not be able to perform both functions. To address this problem, ADAP has an additional input V_{SHIFT}.

Whenever V_{SHIFT} is asserted for an instruction cycle, the analog storage unit loads a new piece of data from its input. If V_{SHIFT} is not asserted, then the analog storage unit holds its current data. To assert V_{SHIFT}, it should be high at the same part of the cycle that V_F is high. This signal is intended to work on a row by row basis, i.e., each cell in a row will either hold or shift. This is shown in Figure B.1.

If there are not enough pins for every row to have a shift control signal, another way to address the problem is to add an extra bit to the cell control register. If this bit is high, the analog storage unit always loads a new value. If the bit is low, the analog storage unit only loads a new value if V_{SHIFT} was high ($V_{SHIFT,local} = V_{SHIFT,global} + Control_{B}it$). This would add extra area to the cell register, but add flexibility and cut down on the pin count.

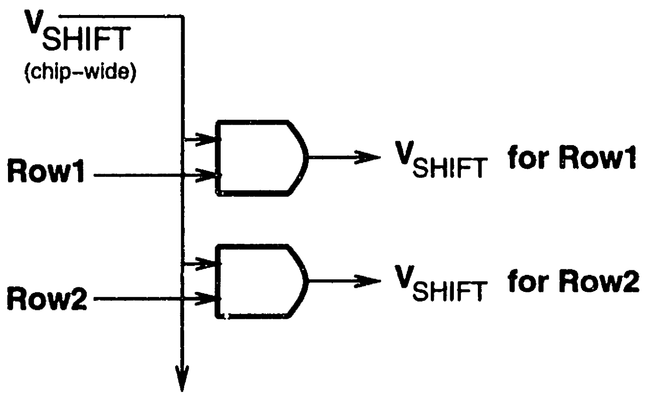


Figure B.1: Current V_{SHIFT} Circuit

B.4 Digital Buffer Timing

The timing of the shift register used in the digital buffer is shown in Figure B.2. When $\overline{\phi_{12}}$ goes high, the bit V_{IN} is loaded into the register and appears at the output V_{OUT} after a delay. The output of the register is then valid while $\overline{\phi_{12}}$ is low.

When $\overline{\phi_{12}}$ is low, the data is stored on the gate capacitance of M11 and M12. This is sufficient for the speed at which the shift register is clocked, but it would not be suitable for long term storage of the information.

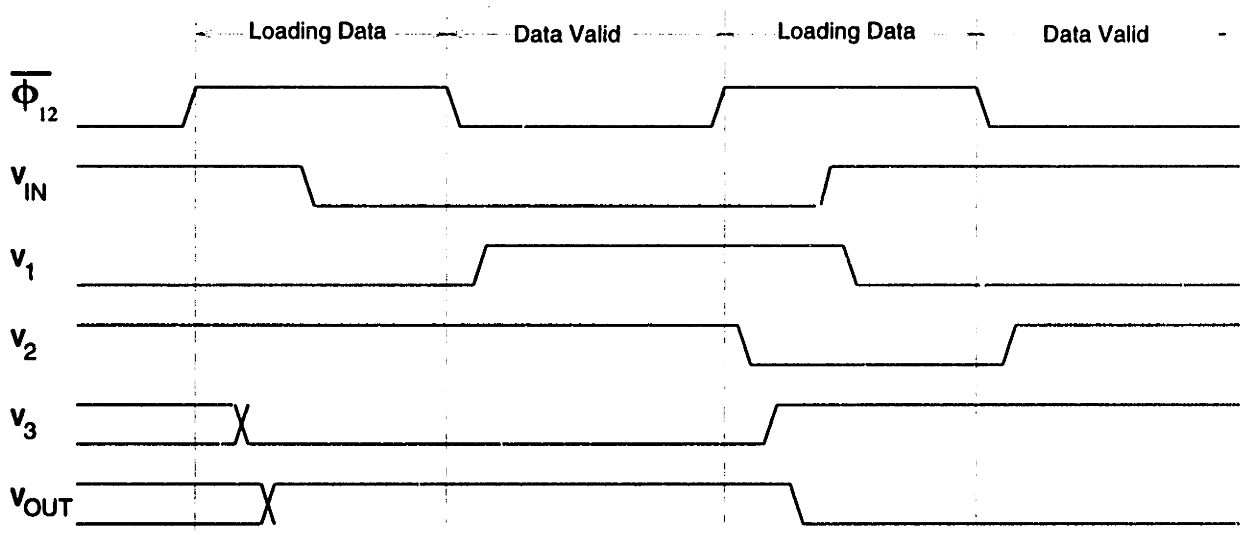


Figure B.2: Shift Register Timing

There are two potential timing problems with the register circuit shown in Figure B.3 which are avoided by proper transistor sizing. The first timing problem occurs when $\overline{\phi_{12}}$ goes low. When this happens, V_2 goes high. It is important that M9 turn off before V_2 goes high, otherwise V_3 may change its value. To ensure this, M7 is made weak so that it charges

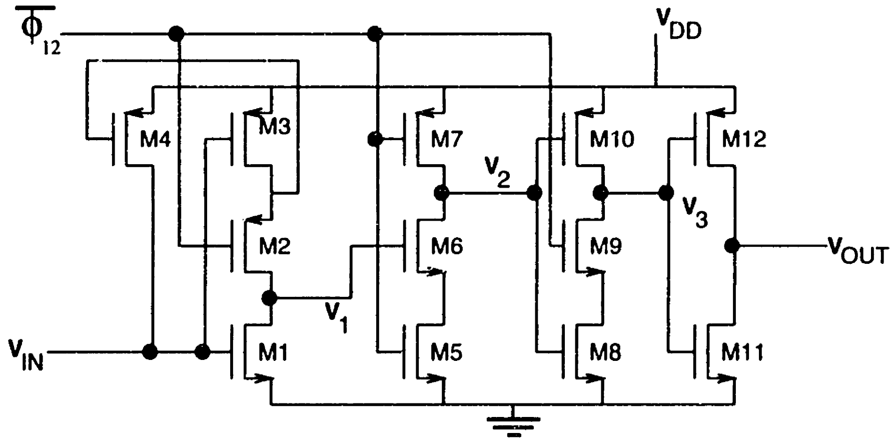


Figure B.3: Shift Register Schematic

up V_2 slowly. The second potential problem occurs if V_{IN} goes from low to high while $\overline{\phi_{12}}$ is high. This will cause V_1 to go low. It is important that this happen after V_1 has caused V_2 to go low, otherwise V_2 will stay high, which will be incorrect. To ensure this, M1 is made weak so that it discharges V_1 slowly.

B.5 Clock Generation

The schematic of the Clock Generator Board is shown in Figure B.4. The inverters used to generate ϕ_1 and ϕ_2 can have capacitors added to their outputs to adjust the size of the non-overlapping period. Larger capacitors create larger non-overlapping periods. 74LS163 chips were used to generate $\overline{V_F}$ and $\overline{V_L}$, but any sort of counter circuit, appropriately wired, can be used.

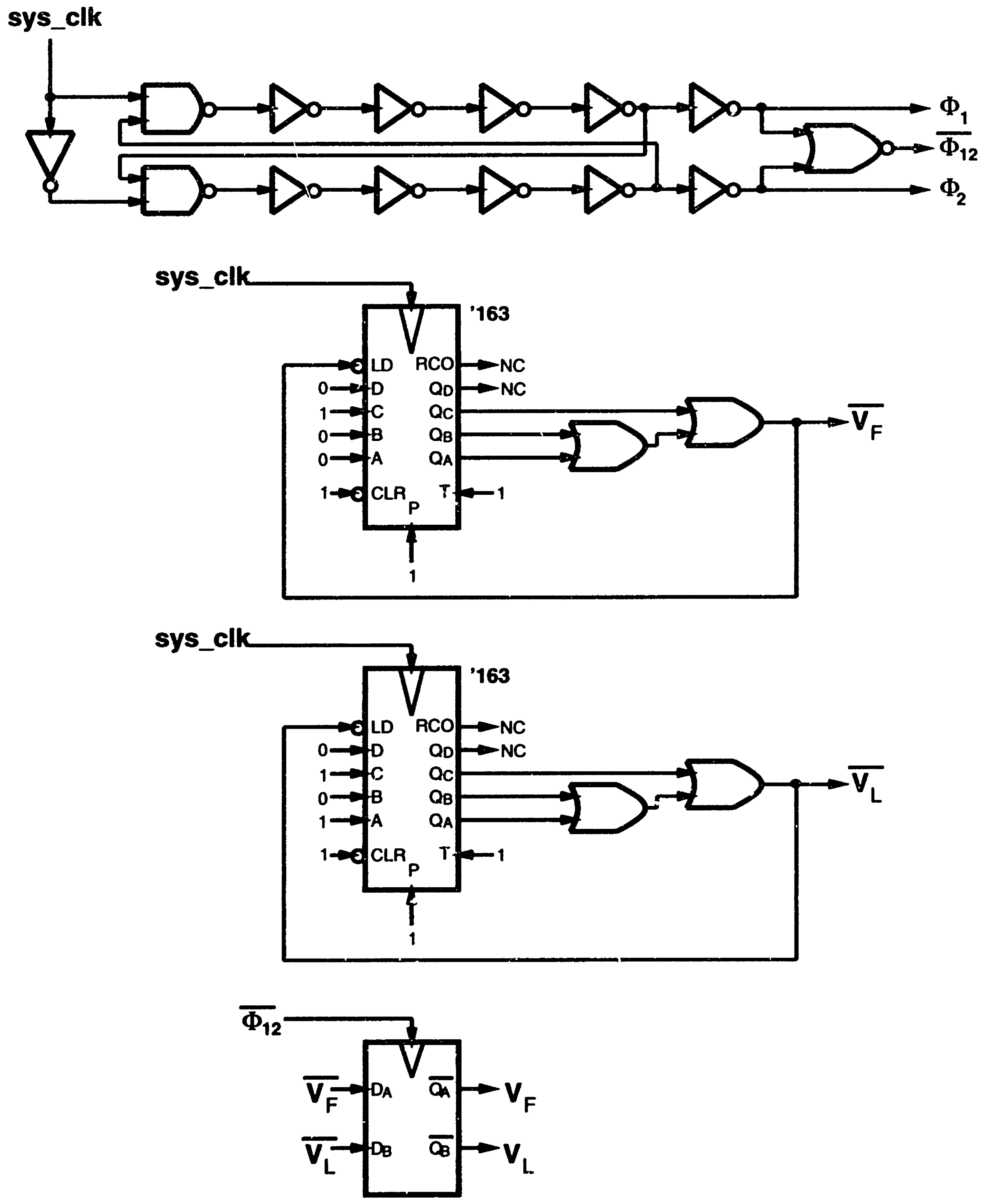


Figure B.4: Schematic of Clock Generator Circuit

Appendix C

Array Programming

The array picture in Figure C.1 shows in graphical form how the array was programmed to perform the edge detection function. K2 was set using the S input, K1 was set using the X input, and D_{N-3} was set using the E2 input. The output was accessed by adding the calculation result to zero and taking the digital output of the middle A/D. The C program used to calculate the edge map on a workstation is:

```
#include <stdio.h>
#include <math.h>

main(argc,argv)
int argc;
char *argv[];
{
  FILE *input_file, *output_file;
  char filename[256];
  double edge1, edge2, edge3;
  int x1,x2,x3,x4,x5;

  strcpy(filename, "pic");
  strcat(filename, argv[1]);
  strcat(filename, ".dat");
  input_file = (FILE *) fopen(filename, "r");

  strcpy(filename, "pic");
  strcat(filename, argv[1]);
  strcat(filename, ".edg");
  output_file = (FILE *) fopen(filename, "w");

  while (fscanf(input_file, "%d %d %d %d %d", &x5,&x4,&x3,&x2,&x1) != EOF) {
    edge2 = ((double) x5) * .6 + ((double) x4) * .6;
    edge1 = ((double) x1) * .12 + ((double) x2) * .12;
    if (edge2 > 255) edge2 = 255;
    if (edge1 > 255) edge1 = 255;
    if (edge1 < 1) edge1 = 1;
    edge3 = 2304/edge1;               /* 2304 = 256*9 */
    if (edge3 > 255) edge3 = 255;
    edge3 = edge2 * edge3/256;
    edge3 = floor(edge3);
    if (edge3 > 255) edge3 = 255;
    fprintf(output_file, "%d\n", (int) edge3);
  }
}
```

The array picture in Figure C.2 shows in graphical form how the array was programmed to perform the sub pixel resolution function. 64 was set using the W input, 128 was set using the X input, 192 was set using the E3 input, and D_{N-2} was set using the E4 input. The output was accessed by adding the calculation result to zero and taking the digital output of the second A/D. The C program used to calculate the sub pixel resolution on a workstation is:

```
#include <stdio.h>
#include <math.h>


main(argc,argv)
int argc;
char *argv[];
{


  FILE *input_file, *output_file;
  char filename[256];
  double edge1, edge2, edge3;
  int x1,x2,x3;

  strcpy(filename, "cpic");
  strcat(filename, argv[1]);
  strcat(filename, ".tad");
  input_file = (FILE *) fopen(filename, "r");

  strcpy(filename, "cpic");
  strcat(filename, argv[1]);
  strcat(filename, ".spx");
  output_file = (FILE *) fopen(filename, "w");

  while (fscanf(input_file, "%d %d %d", &x1,&x2,&x3) != EOF) {
    edge2 = -((double) x1) * .75 - ((double) x3) * .25 + ((double) x2);
    if (edge2 < 0) edge2 = 0;
    edge1 = -((double) x1) * .5 - ((double) x3) * .5 + ((double) x2);
    if (edge1 < 1) edge1 = 1;
    if (edge2 > 255) edge2 = 255;
    if (edge1 > 255) edge1 = 255;
    edge3 = 2304/edge1;              /* 2304 = 256*9 */
    if (edge3 > 255) edge3 = 255;
    edge3 = edge2 * edge3/256;
    if (edge3 > 255) edge3 = 255;
    edge3 = floor(edge3);
    fprintf(output_file, "%f\n", edge3/9);
  }
}
```

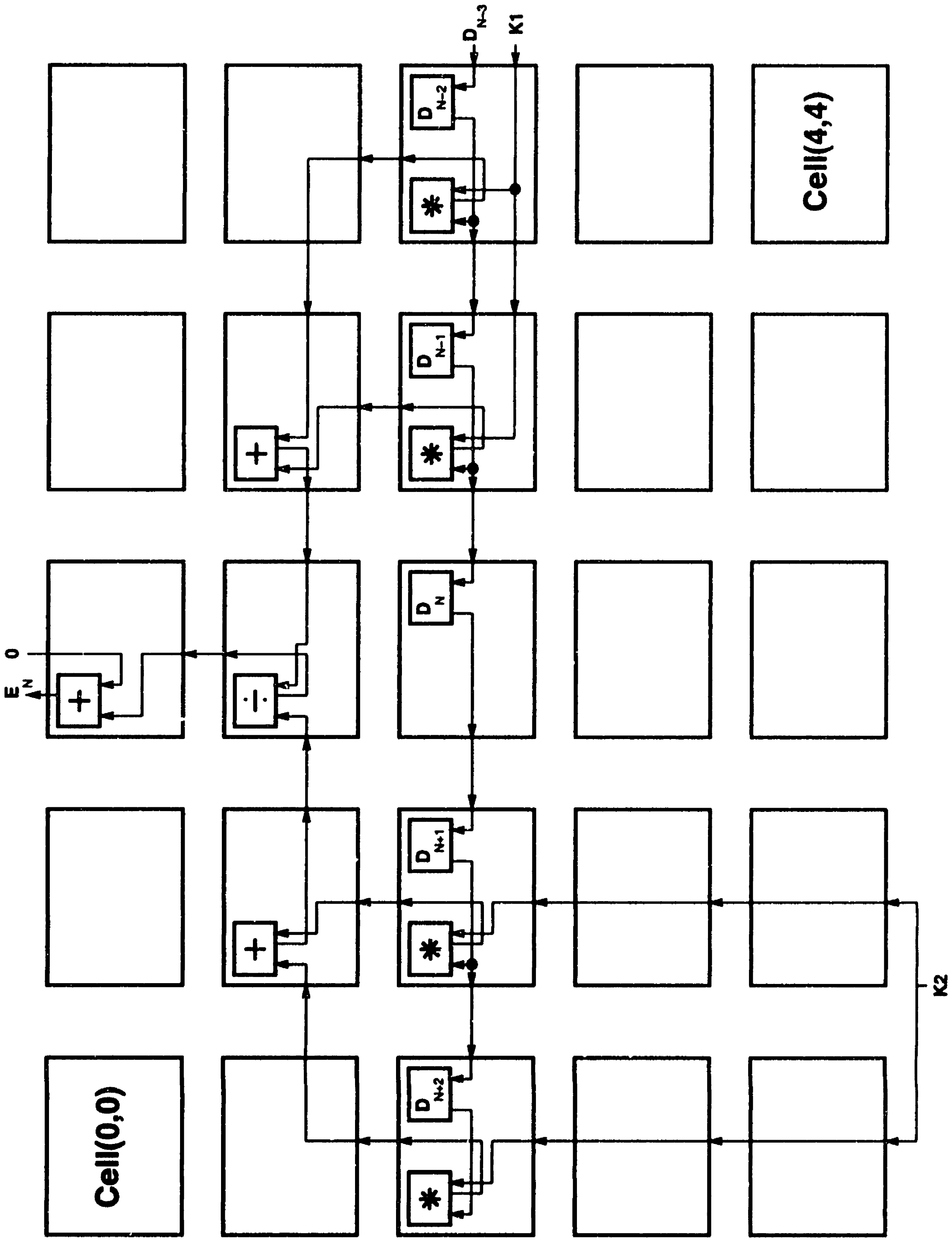


Figure C.1: Array Program for Edge Detection

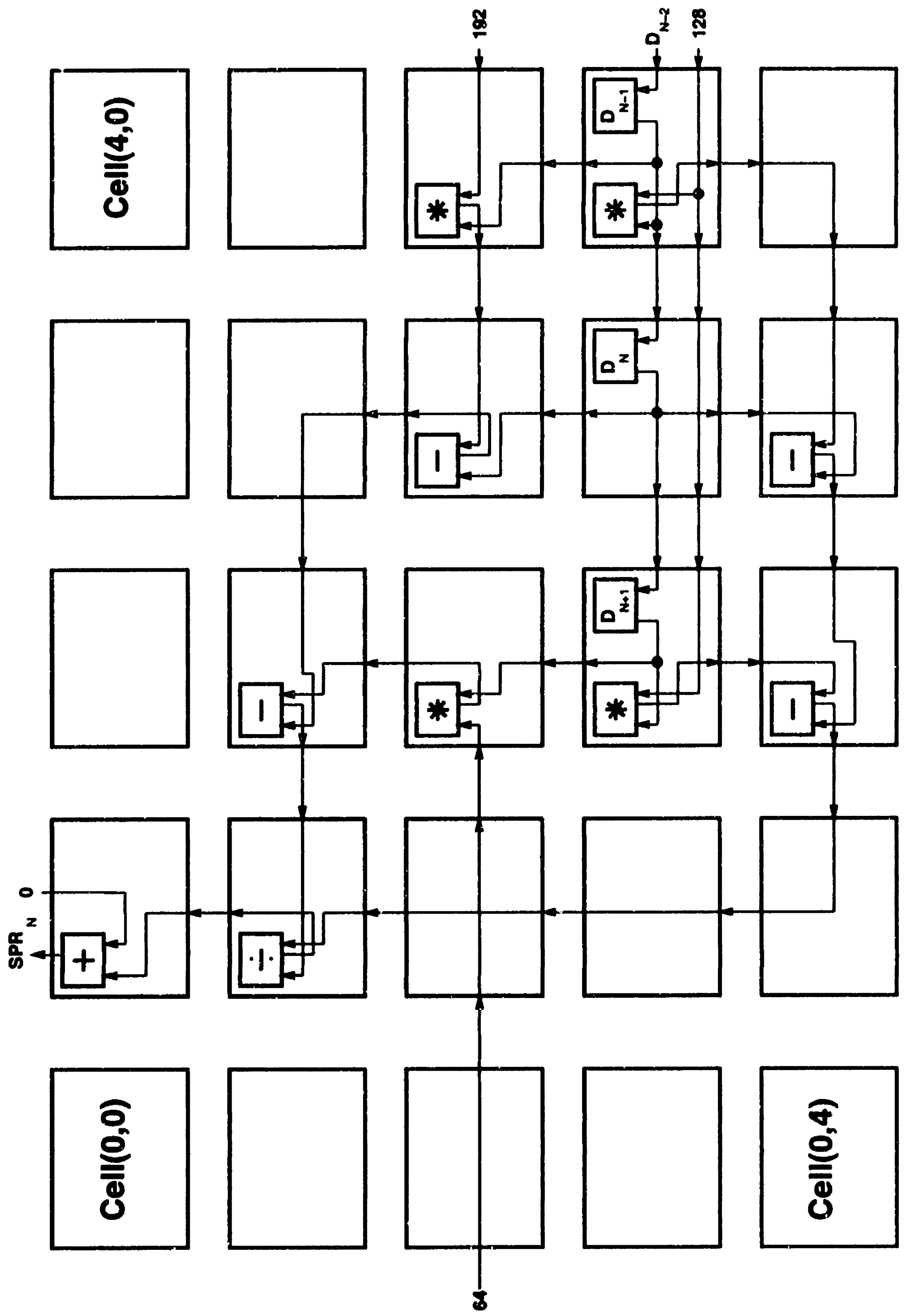


Figure C.2: Array Program for Sub Pixel Resolution

To program the array, a file is created for each processor. This file contains the values for every bit in the control register. An example is shown below:

```
filename: ci132.dat

1:W-A3: 0
2:MI-A3: 0
3:MI-W: 0
4:MO-W: 1
5:MO-N: 0
6:MO-S: 0
7:MO-E: 0
8:MI-E: 1
9:MI-N: 0
10:MI-S: 0
11:N-A3: 1
12:S-A3: 0
13:E-A3: 0
14:*: 1
15:+: 0
16:-: 0
17:/: 0
18:N-E: 0
19:A2-W: 0
20:A2-N: 0
21:A1-S: 0
22:A1-E: 0
23:A1-W: 0
24:A1-X: 1
25:A1-MO: 0
26:A2-MO: 1
27:A1-N: 0
28:A2-S: 0
29:A2-E: 0
30:A1-AI: 0
31:E-W: 0
32:S-W: 0
33:N-S: 0
34:N-W: 0
35:S-E: 0
```

The filename ci132.dat means that it is program number 1, processor cell(3,2). The line number corresponds to the number of the bit in the control register, the text corresponds to th econnection of arithmetic function being set, and the 0 or 1 determines which connection/function is activated. A 0 corresponds to OFF, a 1 corresponds to ON. So, in this example, the analog storage unit is reading data from the east port (8:MI-E:1) and writing data to the west port (4:MO-W:1). The ALU is performing multiplication (14:*:1); its inputs are X (24:A1-X:1) and the output of the analog storage unit(26:A2-MO:1); its output goes to the north port(11:N-A3:1).**It is the programmer's responsibility to make sure that the program makes sense, e.g., only one arithmetic function should be turned on.**

The data from each of these files is then combined into one file which contains the numbers that are loaded into the array. This program will depend on the array structure, the one I used places the data into a file called aiN.dat, where N is the number of the program:

```
/homes/damartin/phd/data/array_alg/array_data.c

/* Program for making datafile for programming the array         */
/* David Martin                                                  */
/* Microsystems Technology Laboratory                            */
/* 04-01-96                                                      */

#include <stdio.h>
#include <malloc.h>
#include <math.h>

#define NUM_OF_BYTES 35 * 5
#define NUM_OF_BITS 35

main(argc, argv)
int argc;
char *argv[];
{

  FILE *file;
  int i,x,y;
  int *data;
  int bi:,
  char filename[256];
  char pointer1[256], pointer2[256];


  data = (int *) calloc(NUM_OF_BYTES, sizeof(int));
  for (i = 0; i < NUM_OF_BYTES; ++i) {
    data[i] = 32;
  }
```

```
    for (y = 0; y <= 4; ++y) {
      for (x = 0; x <= 4; ++x) {
        strcpy(filename, "ci");
        strcat(filename, argv[1]);
        strncat(filename, lltostr( (long long) x, pointer1), 1);
        strncat(filename, lltostr( (long long) y, pointer2), 1);
        strcat(filename, ".dat");
        printf("Reading data file  %s\n", filename);
        file = fopen(filename, "r");
        for (i = 0; i < NUM_OF_BITS; ++i) {
          fscanf(file,"%s %d", pointer1, &bit);
          if (!((bit == 0) || (bit == 1))) {
            printf("Illegal data in %s at line %d.\n", filename,i);
          }
          if (bit == 1) {
            data[x*NUM_OF_BITS + i] = data[x*NUM_OF_BITS + i] | (1  << y);
          }
        }
        fclose(file);
      }
    }

     strcpy (filename, "ai");
     strcat (filename, argv[1]);
     strcat(filename, ".dat");
     printf("Writing to file %s\n", filename);
     file = fopen(filename, "w");
     for(i = 0; i < NUM_OF_BYTES; ++i) {
       fprintf(file, "%d\n", data[i]);
     }
     fclose(file);
  }
```

Appendix D

Array Chip

A 5 by 5 ADAP array chip was fabricated. Figure D.1 is a block diagram of the chip. Bias voltages, clocks, and power signals have been left out of the diagram for clarity. Table D.1 and D.2 provide the pin information for the chip. The chip was packaged in a 65 pin PGA. This is a 1.000" square package with a 0.400" square cavity. The pad to pin mapping is:

```
      K   J   H   G   F   E   D   C   B   A
    +----------------------------------------+
    |                                        |
 1  | 17  15  13  11   9   7   6   4   3   1 | 1
    |                                        |
 2  | 19  16  14  12  10   8   5   2  64  63 | 2
    |                                        |
 3  | 20  18                      65  62  61 | 3
    |                                        |
 4  | 22  21                          60  59 | 4
    |                                        |
 5  | 23  24                          58  57 | 5
    |                                        |
 6  | 25  26                          56  55 | 6
    |                                        |
 7  | 27  28                          53  54 | 7
    |                                        |
 8  | 29  30                          50  52 | 8
    |                                        |
 9  | 31  32  34  37  40  42  44  46  48  51 | 9
    |                                        |
10  | 33  35  36  38  39  41  43  45  47  49 | 10
    |                                        |
    +----------------------------------------+
      K   J   H   G   F   E   D   C   B   A
```

Number	Name	Description	Digital or Analog
1	AI	AI Input	Analog
2	N	North Input/Output	Analog
3	NC	Not Connected	Either
4	VDD_I	Internal Digital VDD	Digital
5	GND_I	Internal Digital GND	Digital
6	GND_I	Internal Digital GND	Digital
7	VDD_I	Internal Digital VDD	Digital
8	GND_O	GND for Output Drivers	Digital
9	D_OUT4	Output of Cell(4,0)	Digital
10	D_OUT3	Output of Cell(3,0)	Digital
11	D_OUT2	Output of Cell(2,0)	Digital
12	VDD_O	VDD for Output Drivers	Digital
13	D_OUT1	Output of Cell(1,0)	Digital
14	VDD_O	VDD for Output Drivers	Digital
15	D_OUT0	Output of Cell(0,0)	Digital
16	GND_O	GND for Output Drivers	Digital
17	GND_B	GND for Input Drivers	Digital
18	VSHIFT	Timing Signal	Digital
19	VF	Timing Signal (First)	Digital
20	VL	Timing Signal (Last)	Digital
21	D_IN0	D_IN for Row 0	Digital
22	D_IN1	D_IN for Row 1	Digital
23	VDD_B	VDD for Input Drivers	Digital
24	D_IN2	D_IN for Row 2	Digital
25	D_IN3	D_IN for Row 3	Digital
26	VDD_B	VDD for Input Drivers	Digital
27	D_IN4	D_IN for Row 4	Digital
28	PHI1	PHI1	Digital
29	PHI2	PHI2	Digital
30	LOAD	LOAD Program Signal	Digital
31	GND_B	GND for Input Drivers	Digital
32	VDD_I	Internal Digital VDD	Digital
33	GND_I	Internal Digital GND	Digital

Table D.1: Array Test Chip Pins

Number	Name	Description	Digital or Analog
34	VDD_A	Internal Analog VDD	Analog
35	GND_A	Internal Analog GND	Analog
36	W	West Input/Output	Analog
37 -45	NC	Not Connected	Either
46	VJ	VJ Bias Voltage	Analog
47	GND_A	Internal Analog GND	Analog
48	S	South Input/Output	Analog
49	VREF2	VREF2 Bias Voltage	Analog
50	VDD_A	Internal Analog VDD	Analog
51	GND_A	Internal Analog GND	Analog
52	VREF	VREF Bias Voltage	Analog
53	VJ	VJ Bias Voltage	Analog
54	E4	East Input/Output for Cell(4,4)	Analog
55	VGPLUS	VGPLUS Bias Voltage	Analog
56	E3	East Input/Output for Cell(4,3)	Analog
57	E2	East Input/Output for Cell(4,2)	Analog
58	X	X Input	Analog
59	E1	East Input/Output for Cell(4,1)	Analog
60	E0	East Input/Output for Cell(4,0)	Analog
61	VG	VG Bias Voltage	Analog
62	VPLUS	VPLUS Voltage	Analog
63	VDD_A	Internal Analog VDD	Analog
64	GND_A	Internal Analog GND	Analog

Table D.2: Array Test Chip Pins

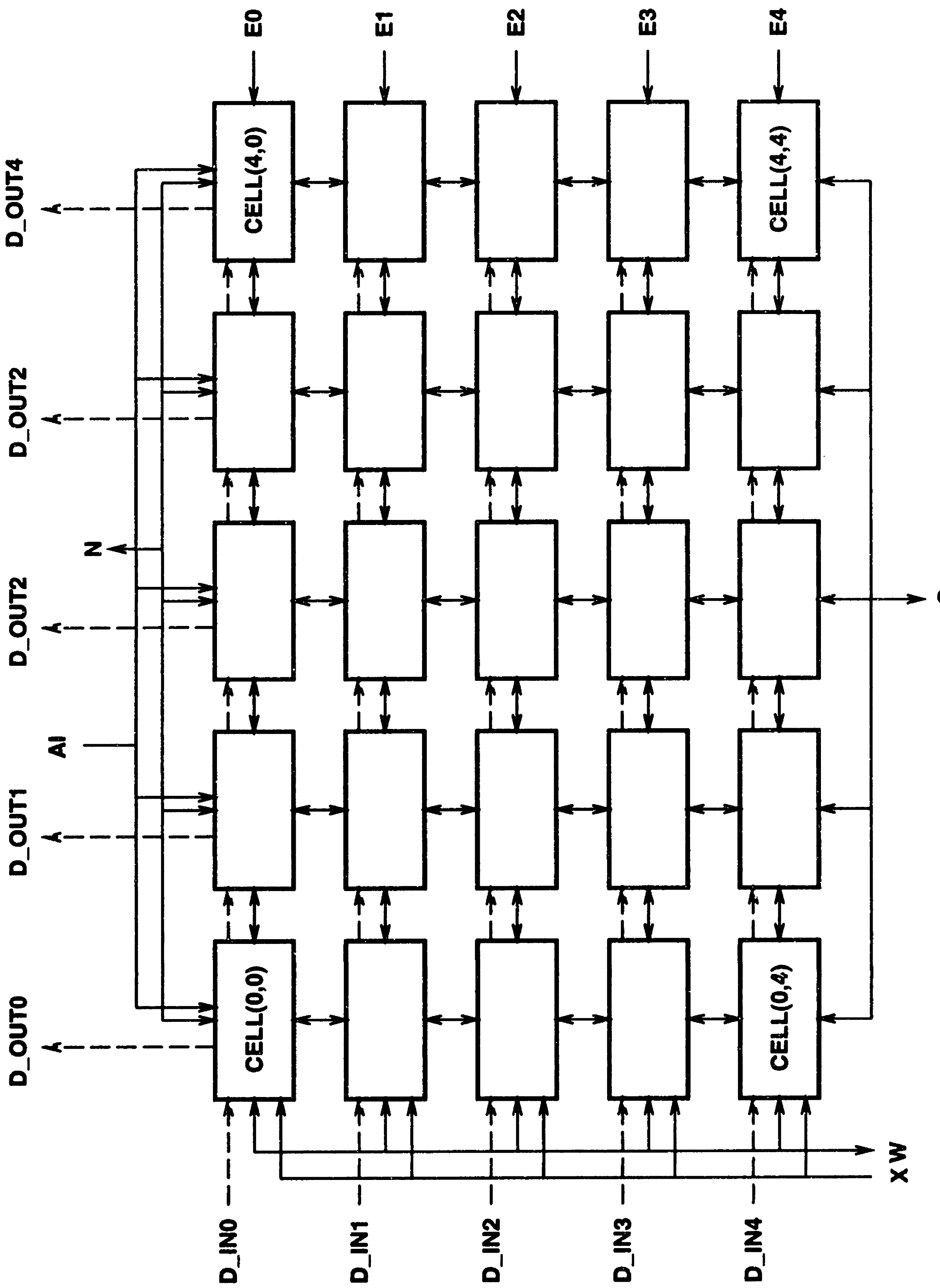


Figure D.1: Array Test Chip Block Diagram

Appendix E

Test Setup Instructions

E.1 Running the Test

The array can be tested by following these instructions:

E.1.1 Convert the data

- Goto the relevant directory on the workstation. For my directory, this is /homes/damartin/phd/da
- Type "convert_picture N", where N is the number of the dataset. This will convert the image "pic$N$.pgm" into the right format for ADAP and put it into a file called "pic$N$.dat". I suggest using $N = 3$ for a slice of the tower picture, or $N = 8$ for the small car picture.

E.1.2 FTP data from the workstation to the PC

The FTP program on the PC does not work very well. It will only work if you follow these instructions **exactly**.

- Goto to the /clarkson directory on the PC.
- Type "termin 0x6a". Ignore any response from the PC.
- Type "3c509 0x6a"
- Type "ftpbin mtl.mit.edu"
- Enter username and password.
- Type "lcd ..
  usr
  david"

- Then cd to the relevant directory on the workstation.
- FTP the data ("pic$N$.dat") from the relevant workstation/directory, where N is the number of the dataset.

E.1.3 Ground Youreslf

E.1.4 Power up Test Setup

Next the chip power supplies are ramped up, and the test board power supplies are turned on.

- Ramp up analog chip supply to 5.1V. This is labeled AVDD. Use the power supply current monitor to monitor the current, and the multimeter to monitor the voltage.
- Ramp up digital chip supply to 5.1V. This is labeled CDVDD. Use the power supply current monitor to monitor the current, and the multimeter to monitor the voltage.
- Turn on Analog Reference Power Supply (+/- 15V, 5V) This is labeled BA5VDD, BAVDD, and BAVSS.
- Turn on Board Digital Power supply (5V). This is labeled ODVDD and BDVDD.
- Turn on the interface board power supply. This is labeled IDVDD.
- Check that the frequency generator output is not disabled.
- Check with the oscilloscope V_F on the Digital Acquisition Board.

E.1.5 Program Array

- Lower the frequency on the frequency generator to 5kHz. This is done not because the array can't be programmed quickly, but because it made the digital interface easy to design. It allows the computer to assume that it is much faster than the array.
- Goto the /usr/david directory on the PC.
- Type "putpro 0" on the PC. This will load program 0 into the array.
- Put the frequency back up to full speed (1MHz or 4MHz, usually).

E.1.6 Check Voltages

- Make sure that the voltage S on the PCB is at 2.667V.
- Make sure that the voltage $E3$ on the PCB is at 1.683V.
- Make sure that S and $E3$ are connected to the ADAP chip with jumpers.

E.1.7 Initialize Data Precision 8100

- Make sure the $E3$ input is disconnected from the Data Precision.
- type "start N" on the PC, where N is the number of the dataset. This will initialize the Data Precision output to the first voltage value. Otherwise, if this was done when it was connected to the chip, it will produce a voltage spike which will damage the chip.
- Verify that the Data Precision is initialized by checking that the word "rem" is lit up on the Data Precision console.
- Connect input $E3$ to the Data Precision

E.1.8 Run Test

Type "acqedg N", where N is the number of the dataset. This runs the test. The computer sends the data to the Data Precision, waits for the Data Precision output to settle, asserts V_F for one instruction cycle to clock the data into the array, and then reads the array output. When the numbers on the Data Precision stop flashing, the test is done.

E.1.9 FTP Data to Workstations

- Goto to the /clarkson directory on the PC.
- Type "termin 0x6a". Ignore any response from the PC.
- Type "3c509 0x6a"
- Type "ftpbin mtl.mit.edu"
- Enter username and password.
- Type "lcd ..
  usr
  david"
- Then cd to the relevant directory on the workstation.
- FTP the data ("pic$N$.edg") to the relevant workstation/directory.

E.1.10 Reconverting the data

- Goto the relevant directory on the workstation.
- Type "reconvert_picture N", where N is the number of the dataset.
- The edge map will be converted to a file named "newpic$N$.pgm", which you can view with xv.

E.2 Files

E.2.1 putpro.c

putpro.c is the program that loads the array program in ai$N$.dat into the array.The code for putpro.c is:

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <dos.h>
#include <stdlib.h>
#include "ieee-c.h"
#include <string.h>
#include <math.h>


/* base address for PXB-721 is 0x300                              */
/* 8255A_3 is presently used                                      */
/* PORT_D3 is control register                                    */

#define OUTPUT_PORT 0x308
#define INPUT_PORT 0x309
#define BYT_CLOCK 0x30A
#define CONFIGURE_PORT 0x30B

/* mode 0x8A configures PORT_A3 as output, PORT_B3 as input      */
/* and PORT_C3 as input   */
#define MODE 0x8B
#define NUMBER_OF_BYTES 35*5
#define PHI1 2
#define PHI2 4

main(argc, argv)
```

```
int argc;
char *argv[];
{
  FILE *file;
  int i, tmp, data[NUMBER_OF_BYTES];
  char filename[256];



  strcpy(filename, "ai");
  strcat(filename, argv[1]);
  strcat(filename, ".dat");
  file = fopen(filename, "r");
  for (i = 0; i < NUMBER_OF_BYTES; ++i) {
    fscanf(file,"%d", &tmp);
    tmp = tmp | 32;
    data[NUMBER_OF_BYTES -1 - i] = tmp;
  }
  fclose(file);

  /* Set up PXB-721 */
  outp(CONFIGURE_PORT, MODE);

/*
 * Send Data to Acquisition Board
 */

  outp(OUTPUT_PORT, data[0]);                /* Load signal up */

  while ((inp(BYT_CLOCK) & PHI2) == 0) ;     /* Get synchronized */
  while ((inp(BYT_CLOCK) & PHI2) != 0) ;
  while ((inp(BYT_CLOCK) & PHI2) == 0) ;
  while ((inp(BYT_CLOCK) & PHI2) != 0) ;
  while ((inp(BYT_CLOCK) & PHI2) == 0) ;

  for (i = 0; i < NUMBER_OF_BYTES; ++i) {
    while((inp(BYT_CLOCK) & PHI2) != 0);
outp(OUTPUT_PORT, data[i]);
  /* printf("%d\n", data[i]); getchar(); */
    while((inp(BYT_CLOCK) & PHI2) == 0) ;
  }
```

```
    while((inp(BYT_CLOCK) & PHI2) != 0);  /* Data latched onto board */
    while((inp(BYT_CLOCK) & PHI2) == 0) ; /* Data latched into register */
    while((inp(BYT_CLOCK) & PHI2) != 0);  /* Data held in register  */
    outp(OUTPUT_PORT, 0);                 /* Load signal down */

}
```

E.2.2 convert_picture.c

convert_picture.c converts the image in pgm format into the format that can be sent to the Data Precision. The code for convert_picture.c is:

```
#include <stdio.h>
#include <math.h>


main(argc,argv)
int argc;
char *argv[];
{


  FILE *input_file, *output_file;
  char filename[256];
  char tmp_string[256];
  char *middle_string, voltage_string[256];
  int width, height, tmp;
  int pointer1, pointer2;
  int x,y,i;
  int *image;
  double voltage, max_voltage, min_voltage;
  max_voltage = 3.483;
  min_voltage = 1.435;

  strcpy(filename, "pic");
  strcat(filename, argv[1]);
  strcat(filename, ".pgm");
  input_file = (FILE *) fopen(filename, "r");

  printf("Reading file %s\n", filename);
  fscanf(input_file, "%s %d %d %d", &tmp_string, &width, &height, &tmp);
  image = (int *) calloc(width*height, sizeof(int));
  for (y = 0; y < height; ++y) {
    for (x = 0; x < width; ++x) {
      fscanf(input_file, "%d", &tmp);
      if (tmp > 255) tmp = 255;
      if (tmp < 0) tmp =0;
      image[y*width+x] = tmp;
    }
  }
  fclose(input_file);

  strcpy(filename, "pic");
  strcat(filename, argv[1]);
  strcat(filename, ".dat");
  output_file = (FILE *) fopen(filename, "w");
  printf("Writing to file %s\n", filename);

   for (y = 0; y < height; ++y) {
    for (x = 2; x < width-2; ++x) {
      for(i =2; i >=-2; --i) {
voltage = (double) image[y*width + x + i]/256.0;
voltage = voltage*(max_voltage - min_voltage) + min_voltage;
voltage = voltage * 10000;
voltage = floor(voltage);

strcpy(voltage_string, "V1+0");
middle_string = (char *) ecvt(voltage, 5, &pointer1, &pointer2);
strcat(voltage_string, middle_string);
strcat(voltage_string, "0");



fprintf(output_file, "%s ", voltage_string);

/* fprintf(output_file, "%d ",image[y*width + x + i]); */

      }
      fprintf(output_file, "\n");
    }
   }

   fclose(output_file);
   free(image);


  strcpy(filename, "pic");
  strcat(filename, argv[1]);
  strcat(filename, ".inf");
  output_file = (FILE *) fopen(filename, "w");
  fprintf(output_file,"%d %d", width-4, height);
  fclose(output_file);

}
```

E.2.3 reconvert_picture.c

reconvert_picture.c takes the edge valuse and converts them into an image in pgm format. The code for reconvert_picture.c is:

```
#include <stdio.h>
#include <math.h>


main(argc,argv)
int argc;
char *argv[];
{


  FILE *input_file, *output_file;
  char filename[256];
  char tmp_string[256];
  int width, height, tmp;
  int x,y,i;
  int *image;

  strcpy(filename, "cpic");
  strcat(filename, argv[1]);
  strcat(filename, ".inf");
  input_file = (FILE *) fopen(filename, "r");

  printf("Reading file %s\n", filename);
  fscanf(input_file, "%d %d", &width, &height);
  strcpy(filename, "cpic");
  strcat(filename, argv[1]);
  strcat(filename, ".edg");
  input_file = (FILE *) fopen(filename, "r");
  printf("Reading file %s\n", filename);
  image = (int *) calloc(width*height, sizeof(int));
  for (y = 0; y < height; ++y) {
    for (x = 0; x < width; ++x) {
      fscanf(input_file, "%d", &tmp);
      tmp = 255 -tmp;
      image[y*width+x] = tmp;
    }
  }
  fclose(input_file);


  strcpy(filename, "newcpic");
  strcat(filename, argv[1]);
  strcat(filename, ".pgm");
  output_file = (FILE *) fopen(filename, "w");
  printf("Writing to file %s\n", filename);
  fprintf(output_file, "P2 %d %d 255\n", width, height);

   for (y = 0; y < height; ++y) {
    for (x = 0; x < width; ++x) {
      fprintf(output_file, "%d ", image[y*width + x]);
      }
    fprintf(output_file, "\n");
   }
   fclose(output_file);
   free(image);
}
```

E.2.4 acqedg.c

acqedg.c takes the image data from pic*N*.dat and sends it to the Data Precision. At the same time, it reads the output of the array (the edge values) and stores them in pic*N*.edg. The code for acqedg.c is:

```
#include <stdio.h>
#include <conio.h>
```

```
#include <malloc.h>
#include <dos.h>
#include <stdlib.h>
#include "ieee-c.h"
#include <string.h>
#include <math.h>


/* base address for PXB-721 is 0x300                                  */
/* 8255A_3 is presently used                                          */
/* PORT_D3 is control register                                        */

#define OUTPUT_PORT 0x308
#define INPUT_PORT 0x309
#define BYT_CLOCK 0x30A
#define CONFIGURE_PORT 0x30B

/* mode 0x8A configures PORT_A3 as output, PORT_B3 as input          */
/* and PORT_C3 as input   */
#define MODE 0x8B
#define NUMBER_OF_CODES 512

#define PM2525 22 /* or 11 */
#define DP8200 6
double read_voltage(void);

main(argc, argv)
int argc;
char *argv[];
{
  FILE *input_file, *output_file;
  int raw_data, final_data;
  int status, i;
  char filename[256], v1[100], v2[100],v3[100],v4[100],v5[100];

  /*
   * Get file pointers
   */

  strcpy(filename, "pic");
  strcat(filename, argv[1]);
  strcat(filename, ".edg");
  output_file = fopen(filename, "w");

  strcpy(filename, "pic");
  strcat(filename, argv[1]);
  strcat(filename, ".dat");
  input_file = fopen(filename, "r");

    /*
     * make PC a controller at address 21
     */
/*
  initialize (21,0);
  getchar();  */

  /*
   * Set up PXB-721
   */

  outp(CONFIGURE_PORT, MODE);

  while (fscanf(input_file, "%s %s %s %s %s", v1, v2, v3, v4, v5) != EOF) {

    /*
     * Send voltages to Data Precision.
     */
 send (DP8200, v1, &status);  /*  send voltgae to dp8200 */
 delay(3);
 outp(OUTPUT_PORT, 1);        /* shift it in */
 outp(OUTPUT_PORT, 0);

 send (DP8200, v2, &status);  /*  send voltgae to dp8200 */
 delay(3);
 outp(OUTPUT_PORT, 1);        /* shift it in */
 outp(OUTPUT_PORT, 0);

 send (DP8200, v3, &status);  /*  send voltgae to dp8200 */
 delay(3);
 outp(OUTPUT_PORT, 1);        /* shift it in */
 outp(OUTPUT_PORT, 0);

 send (DP8200, v4, &status);  /*  send voltgae to dp8200 */
 delay(3);
 outp(OUTPUT_PORT, 1);        /* shift it in */
 outp(OUTPUT_PORT, 0);

 send (DP8200, v5, &status);  /*  send voltgae to dp8200 */
 delay(3);
 outp(OUTPUT_PORT, 1);        /* shift it in */
 outp(OUTPUT_PORT, 0);

 /*
  * Get the input.
  */

 delay(2);
 while((inp(BYT_CLOCK) & 1) == 0) ;
 while ((inp(BYT_CLOCK) & 1) == 1) ;
 raw_data = (int)  inp(INPUT_PORT);

 final_data = 0;
 if ((raw_data & 1) != 0) final_data = final_data + 128;
 if ((raw_data & 2) != 0) final_data = final_data + 64;
 if ((raw_data & 4) != 0) final_data = final_data + 32;
 if ((raw_data & 8) != 0) final_data = final_data + 16;
 if ((raw_data & 16) != 0) final_data = final_data + 8;
 if ((raw_data & 32) != 0) final_data = final_data + 4;
 if ((raw_data & 64) != 0) final_data = final_data + 2;
 if ((raw_data & 128) != 0) final_data = final_data + 1;

/*
 * Put data in a file
 */
fprintf(output_file, "%d\n", final_data);
  }
  fclose(input_file);
  fclose(output_file);
}
```

Bibliography

[1] I. S. McQuirk, *An Analog VLSI Chip for Estimating the Focus of Expansion.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1996.

[2] J. M. Hakkarainen, *A Real-Time Stereo Vision System in CCD/CMOS Technology.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1992.

[3] C. L. Keast, *An Integrated Image Acquisition, Smoothing and Segmentation Focal Plane Processor.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1992.

[4] P. C. Yu and H.-S. Lee, "A CMOS Resistive-fuse Processor for 2-D Image Acquisition, Smoothing, and Segmentation," in *Eighteenth European Solid State Circuit Conference*, (Copenhagen, Denmark), 1992.

[5] C. Koch and B. Mathur, "Neuromorphic vision chips," *IEEE Spectrum*, vol. 33, no. 5, pp. 38–46, May 1996.

[6] D. Standley, *Analog VLSI Implementation of Smart Vision Sensors: Stability Theory and an Experimental Design.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1991.

[7] B. Ulmer, "Vita ii- active collision avoidance in real traffic," in *Intelligent Vehicles '94*, pp. 1–6, IEEE Industrial Electronics Society, October 1994.

[8] E.D. Dickmanns,*et al.*, "The seeing passenger car 'vamors-p'," in *Intelligent Vehicles '94*, pp. 68–73, IEEE Industrial Electronics Society, October 1994.

[9] Takashi Gomi,*et al.*, "The development of a fully autonomous ground vehicle (fagv)," in *Intelligent Vehicles '94*, pp. 62–67, IEEE Industrial Electronics Society, October 1994.

[10] J. E. Halpert, "Where the radar meets the road," *The New York Times*, October 16, 1994.

[11] Paul Ganci,*et al.*, "A forward looking automotive radar sensor," in *Intelligent Vehicles '95*, pp. 321–325, IEEE Industrial Electronics Society, September 1995.

[12] Yukinori Yamada,*et al.*, "Development of a 60ghz radar for rear-end collision avoidance," in *Intelligent Vehicles '94*, pp. 207–212, IEEE Industrial Electronics Society, October 1994.

[13] N. Templin, "In the lab: Vision systems in cars of future may ease drivers' fears of dark," *The Wall Street Journal*, September 8, 1994.

[14] K. Saneyoshi, "3-d image recognition system by means of stereoscopy combined with ordinary image processing," in *Intelligent Vehicles '94*, pp. 13–18, IEEE Industrial Electronics Society, October 1994.

[15] S. Raboisson and P. Schmouker, "Obstacle detection in highway environment by colour ccd camera and image processing prototype installed in a vehicle," in *Intelligent Vehicles '94*, pp. 44–49, IEEE Industrial Electronics Society, October 1994.

[16] Thomas Zielke,*et al.*, "Intensity and edge-based symmetry detection applied to car-following," in *Proceedings of ECCV-92*, pp. 865–873, 1992.

[17] Yoshiki Ninomiya,*et al.*, "A real-time vision for intelligent vehicles," in *Intelligent Vehicles '95*, pp. 315–320, IEEE Industrial Electronics Society, September 1995.

[18] Stefan Bohrer,*et al.*, "An integrated obstacle detection framework for intelligent cruise control on motorways," in *Intelligent Vehicles '95*, pp. 276–281, IEEE Industrial Electronics Society, September 1995.

[19] J.-C. Burie and J. Postaire, "Real-time stereo vision with linear cameras," in *Intelligent Vehicles '94*, pp. 369–374, IEEE Industrial Electronics Society, October 1994.

[20] K.I. Kim,*et al.*, "An autonomous land vehicle prv ii: Progress and performance enhancement," in *Intelligent Vehicles '95*, pp. 264–269, IEEE Industrial Electronics Society, September 1995.

[21] D. Koller,*et al.*, "Using binocular stereopsis for vision-based vehicle control," in *Intelligent Vehicles '94*, pp. 237–242, IEEE Industrial Electronics Society, October 1994.

[22] Ichiro Masaki,*et al*, "Cost-efective vision systems for intelligent vehicles," in *Intelligent Vehicles '94*, pp. 39–43, IEEE Industrial Electronics Society, October 1994.

[23] Kuniharu Uchimura,*et al.*, "A high -speed digital neural network chip with low-power chain-reaction architecture," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1862–1867, 1992.

[24] Moritoshi Tasunaga,*et al*, "A self-learning digital neural network using wafer-scale lsi," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 2, pp. 106–113, 1993.

[25] D. Mark Royals, *et al.*, "On the design and implementation of a lossless data compression and decompression chip," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 9, pp. 948–953, 1993.

[26] Shin-ichi Uramoto,*et al.*, "A 100-mhz 2-d discrete cosine transform core processor," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 492–499, 1992.

[27] D. Reuver and H. Klar, "A configurable convolution chip with programmable coefficients," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 7, pp. 1121–1123, 1992.

[28] Nobuyuki Yamashita,*et al.*, "A 3.84 gips integrated memory aray processor with 64 processing elements and a 2-mb sram," *IEEE Journal of Solid State Circuits*, vol. 29, no. 11, pp. 1336–1343, November 1994.

[29] Chu Phoon Chong,*et al.*, "Image-motion detection using analog vlsi," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 93–96, 1992.

[30] I. S. McQuirk, "Direct methods for estimating the focus of expansion in aanlog vlsi," Master's thesis, MIT, September 1991.

[31] Yutaka Arima,*et al*, "A refreshable analog vlsi neural network chip with 400 neurons and 40k synapses," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1854–1861, December 1992.

[32] Bernabé Linares-Barranco,*et al.*, "A modular t-mode design approach for analog neural network hardware implementations," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 5, pp. 701–713, May 1992.

[33] T. Morie and Y. Amemiya, "An all-analog expandable neural network lsi with on-chip backpropagation learning," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 9, pp. 1086–1093, September 1994.

[34] Srinagesh Satyanarayana,*et al.*, "A reconfigurable vlsi neural network," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 67–81, January 1992.

[35] Bernhard E. Boser, *et al.*, "An analog neural network processor with programmable topology," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 12, pp. 2017–2025, 1991.

[36] S. Espejo, *et al.*, "Smart-pixel cellular neural networks in analog current-mode cmos technology," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 8, pp. 895–905, 1994.

[37] Peter Kinget, *et al*, "A programmable analog cellular neural network cmos chip for high speed image processing," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pp. 235–243, 1995.

[38] P. Kinget and M. Steyaert, "An analog parallel array processor for real-time sensor signal processing," in *ISSCC 1996 Digest of Technical Papers*, pp. 92–93, IEEE International Solid-State Circuits Conference, February 1996.

[39] G. Erten and R. Goodman, "Analog vlsi implementation for stereo correspondence between 2-d images," *IEEE Transactions on Neural Networks*, vol. 7, no. 2, pp. 266–277, March 1996.

[40] Yang Ni, *et al.*, "Investigation on an analog stereo retina for automobile applications," in *Intelligent Vehicles '94*, pp. 320–325, IEEE Industrial Electronics Society, October 1994.

[41] J. C. Gealow, J.C., *et al*, "System design for pixel-parallel image processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 1, pp. 32–41, March 1996.

[42] B. K. P. Horn, *Robot Vision.* Cambridge, MA: MIT Press, 1986.

[43] J. C. Russ, *The Image processing Handbook.* Boca Raton, FL: CRC Press, 1995.

[44] Ichiro Masaki,*et al.*, "New architecture paradigms for analog vlsi chips," in *Vision Chips - Implementing Vision Algorithms with Analog VLSI Circuits* (C. Koch and H. Li, eds.), pp. 353–375, IEEE Computer Society Press, 1995.

[45] L. Kaminski, "A sub-pixel stereo vision system for cost-effective intelligent vehicle applications," in *Intelligent Vehicles '95 Symposium*, pp. 7–12, IEEE Industrial Electronics Society, 1995.

[46] D. Naidu and R. Fisher, "A comparison of algorithms for sub-pixel peak detection," Tech. Rep. 553, University of Edinburgh Department of Artificial Intelligence, 1991.

[47] P. R. Gray and R. G. Meyer, *Analysis and Design of Analog Integrated Circuits.* New York: John Wiley & Sons, Inc., 2nd ed., 1984.

[48] F. Forti and M. Wright, "Measurement of MOS Current Mismatch in the Weak Inversion Region," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 2, pp. 138–142, 1994.

[49] J. L. Hennessy and D. A. Patterson, *Computer Organization and Design: The Hardware/Software Interface.* San Francisco: Morgan Kaufmann Publishers, Inc., 1994.

[50] H. S. Lee, "6.775 notes: Advanced a/d conversion techniques," 1990.

[51] J. Yuan and C. Svenson, "High-speed cmos circuit technique," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 1, pp. 62–70, February 1989.

[52] Koichiro Ishibashi, *et al.*, "A 12.5-ns 16-mb cmos sram with common-centroid-geometry-layout sense amplifiers," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 4, pp. 411–418, 1994.

[53] A. N. Karanicolas, *Digital Self-Calibration Techniques for High-Accuracy, High-Speed Analog-to-Digital Converters.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1994.

[54] R. M. Stitt, *Burr-Brown IC Applications Handbook.* Tuscon, AZ: Burr-Brown Corporation, 1994.

[55] B. D. Signore and S. Harris, *Crsytal Semiconductor Corporation Volume 1 Data Book A/D Conversion IC's.* Austin, TX: Crystal Semiconductor Corporation, 1992.

[56] J. Doernberg, H.-S. Lee, and D. Hodges, "Full-speed testing of a/d converters," *IEEE Journal of Solid-State Circuits*, vol. SC-19, no. 6, pp. 820–827, December 1984.

[57] Alan V. Oppenheim and Ronald W. Schafer, *Discrete-Time Signal Processing.* Englewood Cliffs, NJ: Prentice Hall, 1989.

[58] D. Shih, "A Data Path for a Pixel-Parallel Image Processing System," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1995.