MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo 869 May 1981

A Vision Chip

by John Batali

Abstract: The time has come for workers in artificial intelligence to begin building hardware. The theories and algorithms being proposed in AI exceed the capabilities of standard computers. Also, the understanding gained in the hardware implementation of a theory is probably not available any other way.

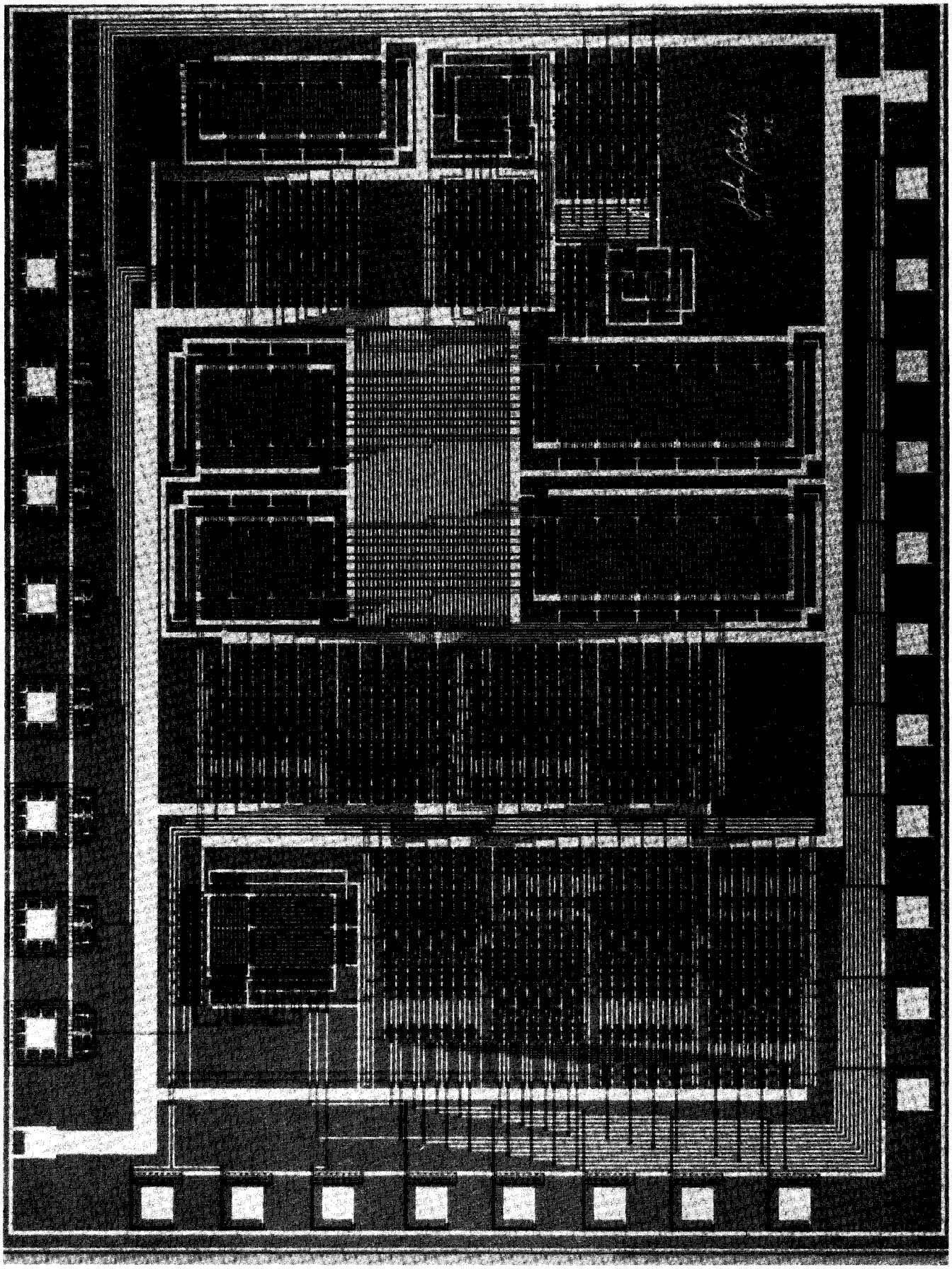
The field of vision is one where progress awaits the speed that hardware implementations can provide. Some well understood and well justified algorithms for early visual processing must be implemented in hardware for later visual processing to be studied.

This paper describes the design and hardware implementation of a particular operator of visual processing. I constructed an NMOS VLSI circuit that computes the gradient, and detects zero-crossings, in a a digital video image in real time. The algorithms employed by the chip, the design process that led to it, and its capabilities and limitations are discussed.

The most difficult aspect of the construction of my vision chip was the attention that had to be paid to very low level detail. For hardware to be a useful tool for AI, designing it must be as much like programming as possible. This paper concludes with some discussion of how such a goal can be met.

This report describes research done at the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505. Support for the Laboratory's VLSI research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contracts N00014-75-C-0643 and N0014-80-C-0622.

This empty page was substituted for a blank page in the original document.



This empty page was substituted for a blank page in the original document.

A Vision Chip

by John Dean Batali

Submitted to the Department of Electrical Engineering
and Computer Science on May 8, 1981
in partial fulfillment of the requirements
for the Degree of Master of Science.

Abstract

The time has come for workers in Artificial Intelligence to begin building hardware. The theories and algorithms being proposed in AI exceed the capabilities of standard computers. Also, the understanding gained in the hardware implementation of a theory is probably not available any other way.

The field of vision is one where progress awaits the speed that hardware implementations can provide. Some well understood and well justified algorithms for early visual processing must be implemented in hardware for later visual processing to be studied.

This paper describes the design and hardware implementation of a particular operator of visual processing. I constructed an NMOS VLSI circuit that computes the gradient, and detects zero-crossings, in a digital video image in real time. The algorithms employed by the chip, the design process that led to it, and its capabilities and limitations are discussed.

The most difficult aspect of the construction of my vision chip was the attention that had to be paid to very low level detail. For hardware to be a useful tool for AI, designing it must be as much like programming as possible. This paper concludes with some discussion of how such a goal can be met.

Thesis Supervisor: Gerald Jay Sussman
Title: Associate Professor of Electrical Engineering and Computer Science

---

Figure 1: (previous page) The DEL chip.

0. INTRODUCTION

Progress to date in AI has depended on the digital computer as a tool and a metaphor. Results have shed light both on the organization of mental processes and the capabilities of digital computers. At the same time, difficulties, problems and new, hard, questions are becoming clear.

In some cases, the problems are not conceptual but computational. The implementation of some proposed, and fairly well understood operators simply takes too long on ordinary computers. It is time to begin the study of the problems associated with special-purpose machines that can be used in AI applications.

One area of AI that could benefit from such an approach is the study of visual perception. The vision problem unites many areas of AI. Somehow the raw information in an intensity array contains a vast amount of meaningful information. Extracting, expressing and using this information is a major goal, and one worthy both of maximum effort and diversity of approach.

It is also an area where computational bottlenecks have appeared. Several approaches to the processing of "low-level" visual information make use of computationally simple operators that perform on a small, localized, area of the intensity array. This paper discusses the implementation of a specific vision operator in VLSI technology. The issues raised range from the reasons why the operator is implementable, through the details of the implementation, to the areas where hardware is useful and the advantages of alternative technologies.

The ability to design working circuits is vital if special purpose machines are to be widely used. However the potential VLSI designer is swamped with considerations ranging from the organization of the algorithm (which he is presumably interested in) to annoying electrical properties of the processing technology (which he certainly is not interested in.) Such a morass of detail is a major obstacle to all but the most stalwart (or foolhardy) AI worker.

But AI in itself is devoted to the study of the management of complicated information in computers. Applying ideas learned from AI research is a plausible way to improve the design process and eventually make it feasible for virtually any well understood algorithm to be implemented in hardware. In fact, the biggest conceptual

gain in such an enterprise might not be the actual implementation of the hardware, but the understanding of the nature of the theories to the degree necessary for their implementation.

VLSI offers to computer science an important chance to experiment with alternative architectures. The capabilities of the new technology far exceed the understanding of those who will program it. The way to begin filling out that understanding is to construct and test special-purpose hardware. The field of vision offers well-understood algorithms that could benefit enormously from hardware -- and the implementation of these operators will provide computer science with some examples of what this new technology can do.

This project is a step into the new arena. I constructed a chip that performs an important low-level vision algorithm. The chip works, but the more important results of this research are the understanding gained about the capabilities of special-purpose VLSI; and the location and elucidation of important difficulties in the design process.

Outline

This paper consists of four chapters. In chapter one, I discuss some of the recent work in the study of visual processing and the reasons that it is both possible and important to consider hardware implementation. In chapter two, I discuss the capabilities that hardware can bring to AI research and the sorts of operations best suited to hardware implementation. I also present the basic architecture of a proposed "vision machine" that is meant to implement some important low-level vision operators. Chapter three is devoted to a description and discussion of the circuit I built. Details of the algorithms and construction are presented. Some performance measures of the chip are given. The chapter concludes with a criticism of the design and suggestions for improvements and alternate approaches. Chapter four is devoted to issues relating to VLSI design. The chapter begins with a description of the design process I used. I then discuss some of the tools that I used, and some that I wished I had used.

Invitation

Most of the work in this project was in the actual design of the chip. The reader

is invited to examine figure 1 and appreciate the magnitude of the effort.

Acknowledgments

I wish to thank IBM for supporting me with a graduate fellowship during part of this research. Neil Mayle and Jack Holloway provided ideas and technical advice. Clark Baker and Chris Terman were very helpful during the verification and simulation of my design; Jon Taft helped me test the finished chip. Gerald Sussman, Clark Baker and Mike Brady read early drafts of this paper. Shimon Ullman got all this started by suggesting this project. Gerald Sussman supervised this thesis and made sure I finished. I thank Gianna Sabella for helping me with the figures and *moral* support.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's V.L.S.I. research is is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract number N00014-80-C-0622 and in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-75-C-0643.

1. THE VISION PROBLEM

The "vision problem" has the goal of obtaining meaningful information from visible light. For the purposes of AI, this usually means that a sensor has produced one or several two-dimensional intensity arrays. From the intensity arrays information must be computed about what is really "out there" in the world that could be important for the perceptual system. This computation seems to involve the segmentation, location, recognition and description of objects. It is important to be able to extract the relevant information from the rest of the data, the information corresponding to features in the world from the noise of the sensing process.

1.1 Early Vision

A visual system must be able to separate and condense valuable information as early in the processing of the information as possible. As the processing of the information renders it more "meaningful" and useful to the system, it is also more difficult to deal with large amounts of it, because the later operators are more complicated. The problem of "early vision" is to reduce the sheer quantity of data while increasing the density of important information.

Several lines of evidence suggest that this is done by biological systems. Neurophysiological studies suggest that, as early as the cells of the retina, a large amount of computation is carried out. Early processing transforms the input from the original two-dimensional intensity array into a smaller collection of more meaningful assertions about the intensity variations of the image in space and time. Evidence from psychophysics suggests that such processing takes place in the human visual system as well. It is possible for humans to extract enormous detail from scenes containing very little semantic content. An artist's line drawing of a scene can often convey much information about the scene despite the fact that the drawing contains virtually none of the intensity information in the scene. This fact suggests that the human visual system produces an intermediate representation of images in a form similar to that of a line drawing, and it is from this representation that later information is obtained.

Such evidence suggests the that processing of visual information begins with the extraction of information-rich "feature points" which summarize the form of the

intensity array at certain places. By limiting the description of the scene to information about the local regions of a few points, the processes can achieve the desired summary of the image. One important problem is determining which points to use as feature points, and determining what sorts of information should be attached to them.

The location and description of the feature points should be as automatic and "low level" as possible. It ought to require as little communication with later processes as possible. These requirements come from the need to produce the feature points quickly and reliably. Indeed, the computation of the feature points could be considered to be a part of the imaging process since the information associated with a feature point depends only on the image itself and not on any interpretation of it. The feature points are useful in that they summarize and express the information in a more compact form than the intensity array. Later perceptual processes, attempting to assemble a semantic interpretation of the image, may then use the feature points exclusively, relying on them as summaries of the vast and unmanageable intensity array.

An example of a feature point would be an assertion that a certain position in the visual field contains an intensity change. The feature point would contain a measure of the "sharpness" of the intensity change, as well as the local orientation of the change, and its temporal behavior.

"Early vision" is the name given to the processes that produce the first symbolic descriptions of the image. Feature points are context-free descriptions of local regions of the image. Since the operators required for these processes must deal only with a local region of the image, each may be quite simple. This is fortuitous, because the total amount of processing is huge.

1.2 Zero-Crossings

An obvious candidate for a kind of feature point is the "edge" assertion. Much visual processing seems to operate on edges -- recognition, shape determination, scene segmentation. It is important for an account of visual information processing to explain what properties of the light patterns from the real world correspond to the perception of edges. Intuitively, an edge is seen at those locations in the image where

there is some change in the intensity, color, or texture of the scene. This suggests that edge perception is related to the computation of a spatial derivative. Perhaps edges are seen at those locations in a scene where some derivative-like operator is at a local maximum. The application of any such operator to real image has the effect of filtering the image in some way. The properties of the filter will effect the results obtained from the derivative operation. It is important that the effects don't impair the accuracy of the system.

[Marr & Hildreth, 1979] make a very specific definition of a class of feature points. According to the theory, the original image is convolved with a mask shaped like the Laplacian of a circularly symmetric gaussian distribution: $\nabla^2 G$ (Figure 2). This operation has the effect of smoothing the image and performing a spatial second-derivative operation on it. The gaussian is the optimal filter for this application because it simultaneously minimizes the error in both the spatial and frequency domains. Evidence from both neurophysiology and psychophysics

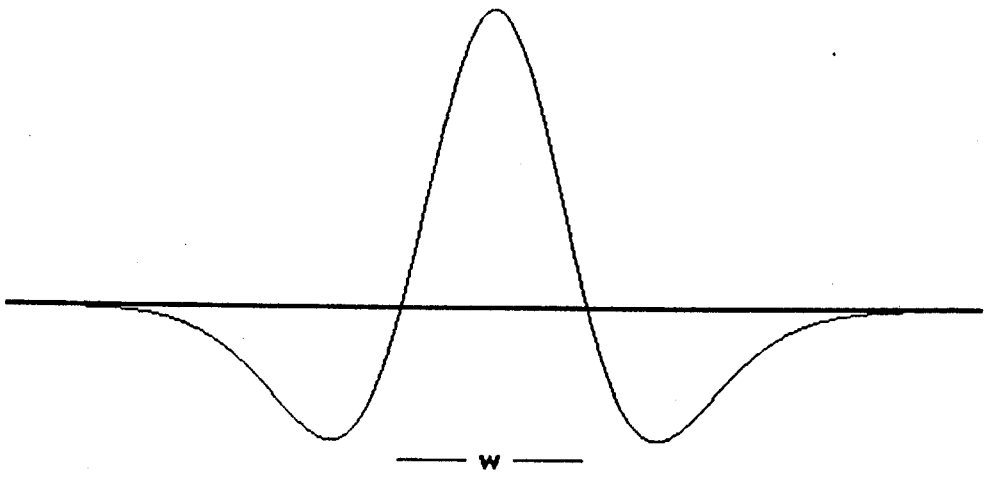


Figure 2: The $\nabla^2 G$ Mask. *The distance* **w**, *the "central panel width" of the mask, is determined by the space constant of the gaussian. The value of* **w** *is related to the scale at which zero-crossings are detected.*

suggests that the filters actually used in mammalian visual systems are very similar to this one.

Points in the convolved image where the values cross zero (zero-crossings) are then located. The zero-crossings represent the locations where the intensity values has maximal spatial changes. Contours of zero-crossings (which roughly follow contours of zero second-derivative) often correspond to the perceived "edges" in an image. The smoothing operation of the gaussian has the effect of band-pass filtering the image in such a way that the zero-crossings locate the edges at a particular spatial scale. The scale at which edges are detected depends on the space constant of the gaussian. The Marr-Hildreth theory suggests that the processing is simultaneously performed on the image with several different mask sizes so that edges may be detected over a range of scales. An example of the results of this process is shown in figure 3.

Information stored with a zero-crossing includes its orientation (the direction of the zero-crossing contour at the location of the particular zero-crossing). The slope of the convolved image, which is related to the contrast of the edge, is represented. A measure of the time-derivative of the image at the point is also represented.

The development of the theory of edge detection was motivated by the need to elucidate a definition of feature points to match in a theory of human stereo vision [Marr & Poggio, 1979]. The problem in stereo perception is to match features in one view of a scene with the "same" features on another view of the same scene. The disparity in the positions of the features may then be used to determine the distance of the object producing the features from the viewing apparatus. Thus the definition of a reliable feature point for the matching operation was vital for the success of the theory.

The stereo theory makes use of zero-crossings detected at the same scale in two views. Properties of the $\nabla^2 G$ mask make it possible to limit the search for a matching zero-crossing from one view in the other. Using a small amount of information about the slope and orientation of the zero-crossings increases the likelihood of a correct match. The matching process begins by using the zero-crossings found in the image after convolution with a large mask. This reduces the number of zero-crossings in the image and makes the problem of finding matches easier. The rough disparity values found this way are then used to constrain the matching process as more accurate

disparity results are obtained from smaller mask sizes. The implementation is discussed in [Grimson, 1980a].

The results of [Logan, 1977] suggest that the zero-crossing description of a band-pass filtered signal is complete in the sense that the original image may be reconstructed from the locations and signs of its zero-crossings. While the filters suggested by the Marr-Hildreth theory do not satisfy Logan's requirements precisely, the result suggests that the zero-crossing description is very rich in information about the original image despite the enormous reduction in data. The work of [Grimson, 1980b] suggests that the zero-crossing description can be used for very precise interpolation of surfaces from the disparity values obtained by stereo matching of zero-crossings.

Zero-crossings were shown to be reliable features for the extraction of motion information from an image [Marr & Ullman, 1980; Batali & Ullman, 1980]. The time derivative of the intensity at the location of a zero-crossing is used to constrain the direction of motion there. Propagation of these constraints over regions of the image allows very accurate determination of the direction of motion of objects and also makes it possible to segment objects moving past one another.

Zero-crossings are used successfully in texture discrimination, surface perception, determination of occluding contour and other visual tasks. [Stevens, 1980; Witkin, 1981; Marr, 1977]

It was learned during the implementation of some of the zero-crossing theories that the auxiliary information (direction and contrast) at the zero-crossings need not be extremely sensitive for the implementations to achieve good performance. The stereo implementation was very successful using only 12 possible values of direction information [Grimson, 1980a] and only the sign of the contrast. The motion work used only 16 possible values for the direction [Batali & Ullman, 1980].

1.3 The Gradient

The gradient operator is a natural choice for early vision, expressing as it does the magnitude of local changes in the intensity of the image. Much work in vision has made use of the gradient. The gradient, together with time-derivative information is

used in the work on optical flow by [Schunk & Horn, 1980] and in the determination of the velocity of moving objects [Fennema & Thompson, 1979].

A problem with the use of the gradient is that it does not reduce the information in the image, but simply transforms it, albeit to a potentially more useful form. However the computation of the gradient is important even for the description of feature points because the gradient of the image at the location of a zero-crossing, expressed in polar form, represents the direction and slope of the zero-crossing. Thus the location of the zero-crossings in an image, together with the computation of the gradient are important early operations that may be performed on an image to locate and describe useful feature points.

2. MACHINES

Properly, the study of AI, indeed all of computer science, is the study of the capabilities and limitations of machines. Indeed the only ultimate proof of the feasibility of the AI enterprise is the existence of a smart machine. Most work to date in AI has consisted of reconfiguring serial machines with programs. As the realization grows that the performance limitations of serial machines are having a significant detrimental effect on the progress of AI, more attention must be given to other architectures.

2.1 Local Processes

As indicated in the previous chapter, much of the work on early visual processing makes use of operators whose computations are extremely "local" in the sense that a particular output datum depends on a strictly spatially limited amount of input data. The convolution, zero-crossing detection, temporal derivative and gradient, may all be reliably determined by using the values of the image intensity in a small region nearby the point of interest.

The suggestion has been made by [Horn, 1973] and [Ullman, 1979], that such local operators are responsible for much of early visual processing. Such operators could be implemented by a large number of simple processes operating in parallel. Parallel operation is vital if the important information is to be extracted in a short period of time. Later processes may be more serial if they have less data to work with.

Other areas of AI make use of programs that suffer from severe bottlenecks in serial machines. In many cases, the desire is to perform a simple symbolic computation on a large number of individuals in a data base. The computations are often as simple as comparing the individuals with some "target". The bottleneck occurs because in serial machines such comparisons, and indeed any processing at all, can only be done in one location. So the amount of time it takes to apply a certain operation to each object in a data-base, no matter how simple the operation, is proportional to the number of items in the data base. This has the effect of slowing performance as the program "knows" more (has a larger data-base), and is certainly not optimal behavior.

Often in such programs, the computation to be performed at each individual depends on very few other individuals. In the simple matching case above, the computation requires only one individual. In cases like this, the nature of the operation to be performed at each individual does not depend on the size of the data-base. As in the case of early vision, the situation consists of a simple process that depends on a small number of elements, working on every element in a large data structure. One could imagine a large number of operators, one per individual in the data-base, all working at the same time, and delivering a result in a constant amount of time. (Actually communications considerations would suggest that the time would depend on the cube root of the number of processes, but the constant of proportionality may be so low -- because the speed of light is so big -- that this wouldn't matter very much.)

2.2 The Need For Hardware

One of the most important parts of the AI methodology is the testing of theories by coding them as computer programs. Many interesting properties of a theory are not apparent until the theory is implemented. The degree to which the implementation agrees with the expected performance can be used to assess the validity of the theory.

If the implementation of a theory is difficult or time-consuming, the theory may not be developed to the degree that would allow a fair assessment. A project whose success depended on the use of an implementation of even a well-understood idea, would, if the implementation were too slow, probably never get adequately tested or exploited.

This is precisely what is happening in vision research. Computing the convolution of a reasonable image (512 by 512 pixels) on a KL-10 computer takes several minutes of CPU time. The actual terminal time on a time-shared system may be measured in hours. Development and implementation of the various zero-crossing theories all depend on the computation of the convolution before any specific further computations may be done. The situation only gets worse as the theories deal with processes that are more and more "high-level." It is easy to lose interest in ideas if testing their validity takes several hours or days.

Other areas of AI face the same problem. Many otherwise interesting projects are never implemented in anything but "toy worlds" because of the time taken to test even the simple cases to be included in theses. A good example is the work of [Doyle, 1980] which contains some well-thought out ideas about problem-solving, knowledge representation and belief justification. The work is the latest on a long evolutionary line of approaches to problem solving and reasoning which have become more and more difficult to implement. And yet implementation of such promising ideas is vital if any progress in AI is to be made. Implementation of these and other ideas is necessary to separate the good, well reasoned ideas from those that are good, well reasoned and work.

The difficulties in the implementation of some AI programs will be alleviated by creating special-purpose hardware for them. In some cases, like the "local" processes mentioned above, the result might be a new, highly parallel, machine. In other cases it might be advantageous to construct a special machine that operates with a serial computer, and is called upon to perform some special task that it can accomplish with blinding speed. (For example: [Rivest, 1980] describes a chip that performs the computations required for an encryption algorithm. The main feature of this chip is a 512-bit ALU. Such a large ALU is necessary for the manipulation of the huge numbers required by the encryption algorithms, but would be rather extravagant for a standard computer.)

2.3 Architectural Theories

AI theories should explicitly deal with the issues relating to their implementations. Just as the AI methodology now requires that theories be expressed as programs, thought should be given to the machines that can run the programs. Some work in this direction has been done, for example [Fahlman, 1979] proposes an abstract machine that implements the important operations in the data-base he proposes for representing knowledge. [Ullman, 1979] suggests certain criteria of "biological feasibility" for proposed brain algorithms. The criteria include a conception of "locality" similar to that above, and attempt to express the sort of algorithms which could be easily implemented in biological systems. I suggest that a notion of "architectural feasibility" be developed and used as a criterion for AI theories.

This suggestion would result in an extension to the role of the computer program in AI. The computer program implementing a process is a proof of the "computational feasibility" of the proposed theory. The details of the implementation or the program are not important to the theory, what is important is the abstractly defined "process" [Marr, 1981]. But just as a program describes a process, it also presupposes an architecture on which the process will run.

In some cases, a serial processor is adequate and might even be shown to be necessary. Even in this case, though, the process might not be the instruction-fetch, execute cycle seen in today's machines. One could imagine a machine running something like a TOTE [Miller, *et al*, 1960] cycle as its primitive, or the primitive operation could be a stimulus-response pairing, or, for an AI example, a production system. Though the processing is serial in all these cases, the primitive operations are quite different. Hardware specially designed for the specific requirements of each operation might be able to achieve better performance than a standard computer.

Of course, it is expected that the optimum architecture for certain processes will contain parallelism of various kinds. Some machines may use relaxation methods or propagation of numerical parameters. Others may use a literal implementation of message-passing or semantic nets. Recognizing and exploiting the parallelism will allow greater understanding of the theories.

The test of an AI theory is its performance as a program. If the program performs badly, it must be possible to differentiate between a bad theory and a good theory on an incompatible machine. The architectural description of a theory will serve as more than simply a suggestion for fast ways to implement it. Just as writing a program to implement a theory forces its creator to explicitly address the issues raised by controlling the "process" he is proposing, specifying the architecture will force him to be aware of interactions that might improve, or hinder, the success of the implementation. To be able to specify the topology of a machine, the theorist must completely understand the topology of the problem. This dimension of understanding must be explored.

An important aspect of architectural feasibility is the asymptotic behavior of the memory requirements of a program. A serious criticism of work in AI [Dreyfus, 1979] is that intelligence involves bringing vast amounts of different kinds on information to bear on a problem. AI programs must be able to store, and access comparable

amounts of information, and must do so in extremely efficient ways.

Another criteria for architectural feasibility is the complexity of communication among submodules of a system. Work in the area and time complexity of algorithms implemented in VLSI suggests that the cost of communication is often the limiting factor in a design [Thompson, 1980]. This situation seems similar to the locality requirement in Ullman's biological feasibility requirements and may be due to the same fundamental factors.

2.4 The Vision Machine

The theories of the early processing of visual information due to David Marr and his colleagues are examples of work in which attention is paid both to the proposed process and the required architecture. Many of the theories developed specify architectural detail to the neuronal level, indeed specific empirical predictions for neurophysiological research are made. The theories suggest processes which are especially amenable to hardware implementation.

Operators suggested by the Marr theories take inputs from a limited region of the image. Often the region of the image that will effect the value of an operator at a point is limited to a few nearby pixels. The early operators make use of little, if any, information from the results of later operators. Each operator takes input from earlier operators (or, ultimately the imaging device), computes a set of values, and sends the results to later operators. In many cases the sensitivity of the operators is presumed to be rather low. As mentioned in the last chapter, the zero-crossing theories use only a few bits of direction and magnitude information at each zero-crossing. This means that the operators need not be extremely accurate for the system as a whole to perform well.

These considerations suggest that hardware implementation of some of the process suggested by the Marr theories might be feasible and useful to research concerned with the later processes. The local nature of the operators and the low sensitivity required, suggests that each operator could be relatively simple. Such hardware would be practical as well, considering that it could compute, for example, stereo depth maps, and other descriptions of a scene.

For the very early portions of visual processing, a full set of parallel operators is not necessary. The convolution, zero-crossing detection and description, and perhaps stereo matching and motion detection could be done in real-time on a simple architecture. The implementations of the operators would operate on the image presented to them as a stream of digital video data. (By "real-time" I mean that the image is processed as quickly as it appears. Roughly this means that the the processing for each frame ought to be done in a time comparable to the frame rate of the video stream. The upper bound on acceptable performance would be about 0.1 second per frame.)

In more detail, the architecture of the "vision machine" is as follows. Consider the imaging device to be producing a digital video signal, each frame of which is WxW pixels in size. The resolution is usually 8 bits per pixel. For many applications W is at least 512. This data is sent as a stream, one scan-line at a time, successive scan-lines in order. (*i.e.* non-interleaved).

The hardware implementing the operators take input from this stream (or the outputs of other operators) and send their results to other operators, or perhaps to a display device, or computer memory for later processing. Each of the proposed operators for the early phases of vision works over a small region of the image. If an operator depends on an NxN region of the image to compute its value, N-1 lines of data must be buffered in a "serpentine memory", and the operator must store (N-1)xN values internally, so that it has the values of all the pixels in the region. The operator computes its results from these values. For the system to operate in real-time, the operator must deliver a result as each new set of data is fed into it -- that is, it must be pipelined. A schematic illustration of the vision machine is shown in figure 4, and the scheme is discussed in [Ullman, 1980].

For an image of 512 by 512 pixels, each operator must compute its results at a rate around 1 Mhz. Such performance is available from several different technologies available today. Some factors influencing the choice of an implementation technology are discussed in section 4.1.

The vision machine architecture as described above is clearly not suited for situations where a large amount of communication is required between processors, or where each processor must be so complicated that it could not operate in a reasonable time. In some vision work involving constraint propagation, such

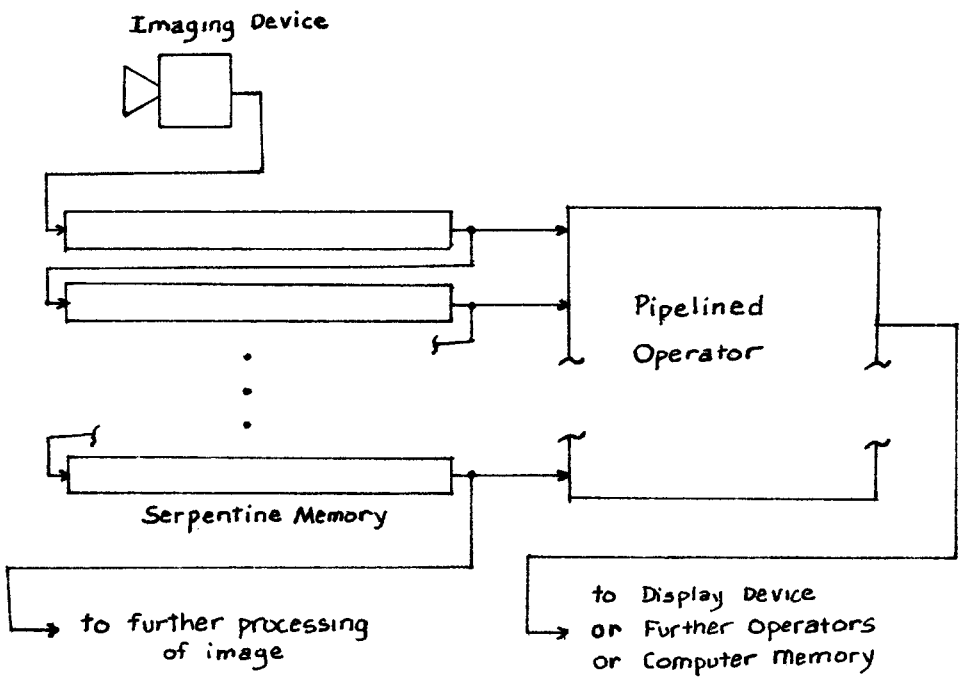


Figure 4: The Vision Machine.

---

communication is required; other vision operators require accurate results from more complicated calculations. In such cases a true multiprocessing arrangement must be used. A system might require a processor for each pixel of the image. If the processors were programmable, it would be possible to use such a system to compute one step of a vision process, save the results at each pixel, reprogram the processors for the next step, and continue the processing. A machine with these capabilities would be useful in other AI applications.

Tradeoffs involved in multiprocessor applications for AI include the relationship

of the sophistication of the processors to the number of them that will be used. For a system of a given complexity, the choice is between using a smaller number of fully-programmable processors with a communications interface; or a large number of very simple processors -- essentially an iterative array, where each element is little more than a finite-state machine.

Another consideration is the imaging device. Present video systems scan the image and produce a digital stream. This representation of the data is well-suited for the vision machine as currently conceived but would be unwieldy for a fully parallel machine. If parallelism is to be employed, it should begin at the imaging device. A scheme to combine the imaging, convolution, and zero-crossing operators in one, highly parallel system, has been proposed by [Tom Knight, personal communication].

3. DEL

In this chapter I present my vision chip, a VLSI integrated circuit for use in the vision machine. It operates on a convolved image and computes the gradient, and indicates the presence of zero-crossings at each point in the image. I call the circuit "DEL." DEL contains roughly 5000 transistors, it is fabricated in silicon-gate NMOS technology and occupies an area of 19.7 square millimeters.

3.1 Reasons

I had two main reasons for pursuing this research. The first was the utility of the result. Research into visual processing has the features that make it an ideal candidate for hardware augmentation: simple, well understood, low-level algorithms that are extremely time-consuming. Other workers [Nishihara & Larson, 1981; Nudd, *et al*, 1979] were developing real-time convolution circuits, the zero-crossing detection step was the logical next point to attack. Such a circuit would speed up a very important pair of intermediate computations which are quite useful in later processing. The success of the stereo and motion theories, together with the extremely large demand they placed on the resources of conventional computers, suggested that continued advances, as well as better understanding of existing theories, would be aided by the faster computation that special purpose devices would make possible.

The other reason was my feeling that hardware will become more and more important to the continued progress of AI. In a sense, acquiring facility with VLSI design is like learning a new programming language. The language is rather limited in expressivity at this point, however it is one of the few concrete representations of parallelism available. Implementing something in hardware forces the designer to become painfully aware of the time, space and interconnectivity requirements of his algorithm. As I discussed in the last chapter, I think that this is a good thing. In any case, I wished to acquire the facility in VLSI design and the project was a means to that goal.

AI in general, and vision in particular, need the computational power that special purpose hardware can provide. VLSI technology offers the capability of producing that hardware. However designing *very* large integrated circuits requires very smart

design tools -- tools that AI can provide. Also, much of what is now called "Computer Science" deals almost exclusively with serial machines. The power that concurrency could bring is not understood. In effect, the power of hardware has passed up our understanding of software -- it seems clear that arbitrarily novel machines could be built, but there are very few candidates. Vision offers several well understood algorithms that lend themselves to straightforward implementations in hardware. Thus this project was an exercise in a new, potentially rich, programming discipline.

Of the two considerations, the latter dominated whenever a conflict arose. That is to say that the primary concern was to learn as much about the LSI design process, and about "programming hardware," as possible, rather than attempting to design a circuit precisely optimal for the job. Specifically, power consumption and speed considerations were not primary in the design, although attempts were made to insure that the circuit would not be grossly suboptimal in either of these regards. Simultaneous with the design of DEL I was involved with the formulation and implementation of various LSI design tools. Lessons learned regarding these will be discussed elsewhere, but I feel that they constitute the most important result of this project.

3.2 Functional Specification

DEL computes an approximation to the two-dimensional gradient, and detects zero-crossings, at each point in a two-dimensional digital video image that has been convolved and is presented to DEL as a raster-scanned digital stream. Each point in the image is an 8-bit 2s-complement intensity value. The actual computation is performed on a 3x3 pixel region of the image. This means that two lines worth of data from the input stream must be buffered in a serpentine memory so as to be presented to DEL in parallel. One of the lines of data only carries the sign of that line, it is used only for the zero-crossing detection. The gradient is computed on a 2x2 pixel region. Thus the serpentine memory requirement is only one complete line of data plus one more line of one bit only. The width of the image doesn't affect the computation performed by DEL, however the size of the serpentine memory does depend on the width of the image.

The chip uses a two-phase non-overlapping clock, and performs its computations in a pipelined manner, with a set of three values taken as input, and an

output value produced each clock cycle. The circuit takes 13 clock cycles to compute its results for each point, it stores intermediate results and buffers them in such a way that all the results for a point -- the zero-crossing indication, direction, and magnitude of the gradient -- appear together 13 clock cycles after the data are presented. No special computations are done at the edges of the image, results computed there will be wrong. DEL is to run at "video rates", meaning that its clock cycle time should be at least one megahertz.

3.3 The Algorithm

A schematic block diagram of the DEL circuit is shown in figure 5. The labeled subsystems will be discussed in this section. In the discussion that follows, we

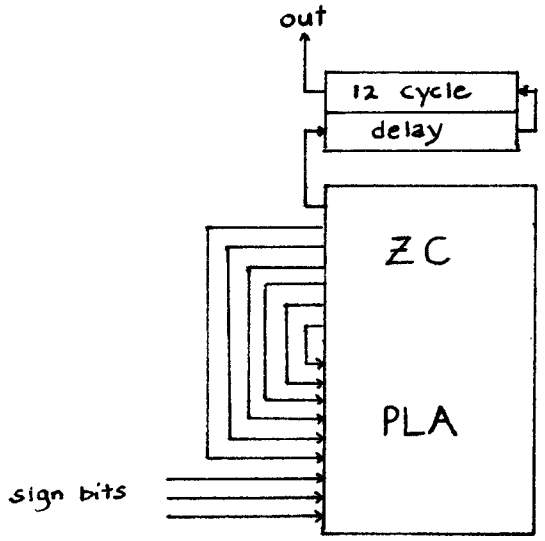


Figure 5a: The Zero-Crossing Detector. *The ZC PLA takes its inputs from the sign bits of the three lines. It recycles the inputs so that each clock cycle it has the sign bits of a 3x3 pixel region. The one bit zero-crossing indication is stored in a 12-cycle shift-register delay to appear at the output at the same time as the gradient information.*

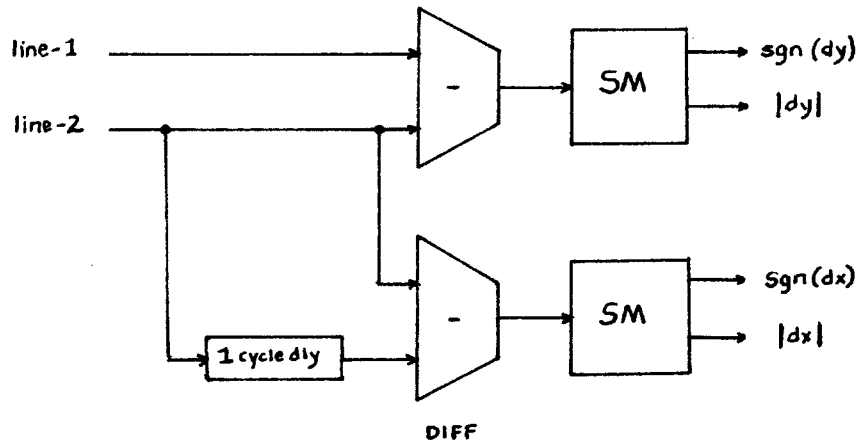


Figure 5b: The DIFF and SM subsystems. *dy is obtained by subtracting the LINE-1 input from the LINE-2 input. dx is obtained by subtracting the current value of the LINE-2 input from the previous one, which has been stored in a 1-bit delay. The SM circuits convert the representation from 2s-complement to sign-magnitude.*

---

consider the operation defined by DEL to be applied to a 3x3 pixel region of an image. As the details of the computation are presented, it will be apparent that the calculation is done in such a way that new data may be accepted each clock cycle, and the results of the inputs of a previous cycle will be output each clock cycle. The input presented each clock cycle consists of the values in a vertical column of three pixels. Conceptually, DEL takes an input of this form each clock cycle and shifts it right the next clock cycle, storing the values in a 3x3 pixel region. The results of the computation done on this region give the values associated with the central pixel of the region. Actually, DEL does not store the values of the entire 3x3 pixel region, but instead keeps only the information it needs for the calculation.

The zero-crossing calculation is performed by the subsystem named ZC by comparing the signs of the values in the 3x3 pixel region with the table shown in figure 6. If the pattern of signs in the image matches one in the table, a zero-crossing

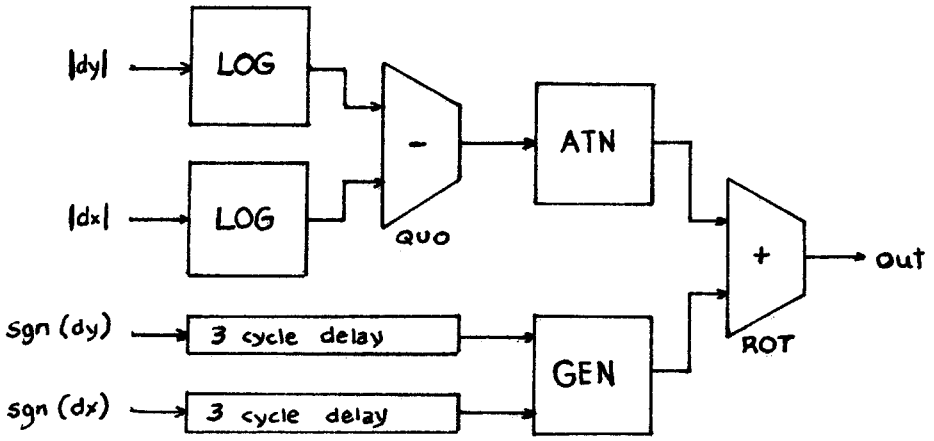


Figure 5c: The Arctangent Computation. *The logarithms of the magnitudes of the differences are subtracted in the QUO subtractor. The difference -- the logarithm of the quotient -- is then given to the ATN table which computes an angle. The signs of the differences are given to the GEN table to determine the appropriate rotation, which is done in the ROT adder.*

---

indication is made for the central point in the region. This indication (a single bit) is then stored in a shift register long enough to be output on the same clock cycle as the gradient values computed for the central point. As indicated in figure 6, a point must fulfill two requirements to be flagged as a zero-crossing: (1) the point must have a non-negative intensity value; (2) one of the neighboring points must be negative.

The gradient is calculated by taking horizontal and vertical first differences across adjacent pixels of the image. This operation is performed by the subsystem DIFF. The vertical differences are taken between the central pixel and the pixel "above" it, the horizontal differences are made between the central pixel and the "next" pixel input. This requires that the central value must be stored internally for one clock cycle. The DIFF subsystem contains an 8-bit shift register for holding the central value for one clock-cycle, and two 8-bit subtractors for performing the

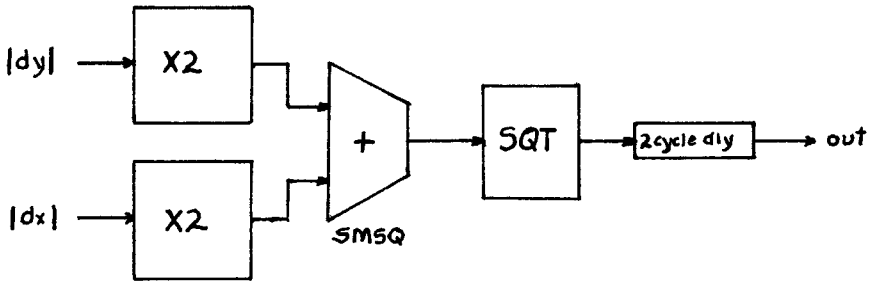


Figure 5d: The Root-Sum-Square computation. *The squares of the magnitudes of the differences are found in the X2 tables. These values are then added in the SMSQ adder and the square root found in the SQT table. The result is then delayed to wait for the arctangent.*

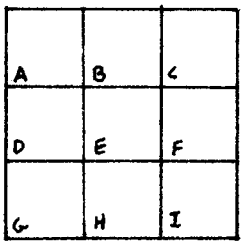
---

differences.

I will call these differences **dx** and **dy**, respectively. The pair **(dx, dy)** is an approximation to the rectangular form of the gradient. This representation is then converted to polar form. The polar form of the gradient is more useful for vision operators because the direction of the gradient along a zero-crossing contour is normal to the contour, pointing to the positive side; the magnitude of the gradient is related to the contrast at the zero-crossing.

Both differences are converted into Sign-Magnitude form in the two SM subsystems. The signs of the differences will not effect the magnitude calculation but will be used later in the calculation of the direction of the gradient.

The direction of the gradient is computed by performing an arctangent operation on the ratio of **dx** and **dy**. The computation is simplified by exploiting the fact that the



sign bit in location	output
A B C D E F G H I	
1 X X X 0 X X X X	1
X 1 X X 0 X X X X	1
X X 1 X 0 X X X X	1
X X X 1 0 X X X X	1
X X X X 1 X X X X	0
X X X X 0 1 X X X	1
X X X X 0 X 1 X X	1
X X X X 0 X X 1 X	1
X X X X 0 X X X 1	1

Figure 6: Zero-Crossing Detection Table. *(Top) The arrangement of pixels. Zero-crossing will be detected at pixel* E. *(Bottom) Programming of ZC PLA.* X *means don't-care.*

---

arctangent of **dy/dx** is the same as the arctangent of **|dy|/|dx|** rotated into the quadrant corresponding to the signs of the differences. (*e.g.* If they are both positive, the result is in the first quadrant.) The quotient operation is performed by computing an approximation to the logarithm of both values and subtracting.

The logarithm approximation is performed by table lookup in the two LOG subsystems. The table is shown in figure 7. Note that the table is based on a priority encoding, only the top 4 bits of the input determine the output. This principle is used in the other large tables in DEL as well.

The logarithms are now subtracted to determine the logarithm of the quotient in

the subsystem QUO, a 4-bit subtractor. The quotient is then fed into the table ATN which computes a 2-bit approximation of the arctangent of the logarithm of its input. Its table is shown in figure 8. This table is quite simple because it is responsible for only two bits of the answer.

The remaining two bits of direction information are computed from the signs of **dx** and **dy** in the system GEN. This is a very simple table that produces the angle by which to rotate the result from ATN to produce the correct angle. The "rotation" is done by the 4-bit adder ROT. The resultant sum is then output as a 4-bit approximation to the direction of the gradient.

Computation of the magnitude of the gradient proceeds in parallel with that of the direction. The magnitude of the gradient is the "Root Sum Square" of the differences: $(dx^2 + dy^2)^{1/2}$. this calculation involves determining an approximation to the square of each input in the X2 tables. The table is presented in figure 9. The sum of the squares is done in the 4-bit SMSQ adder. Finally, the square-root of the sum is computed in the SQT table. Its program is shown in figure 10. The result is then output as a 4-bit approximation to the magnitude of the gradient.

The standard error of the gradient direction calculation, over all input values is 0.644. The standard error of the gradient magnitude calculation is 1.42.

IN	OUT	IN	OUT	IN	OUT
2	2	18	8	64	11
3	3	20	8	72	12
4	4	22	8	80	12
5	4	24	9	88	12
6	5	26	9	96	12
7	5	28	9	104	13
8	6	30	9	112	13
9	6	32	9	120	13
10	6	36	10	128	13
11	6	40	10	144	13
12	7	44	10	160	14
13	7	48	10	176	14
14	7	52	11	192	14
15	7	56	11	208	14
16	8	60	11	224	15
				240	15

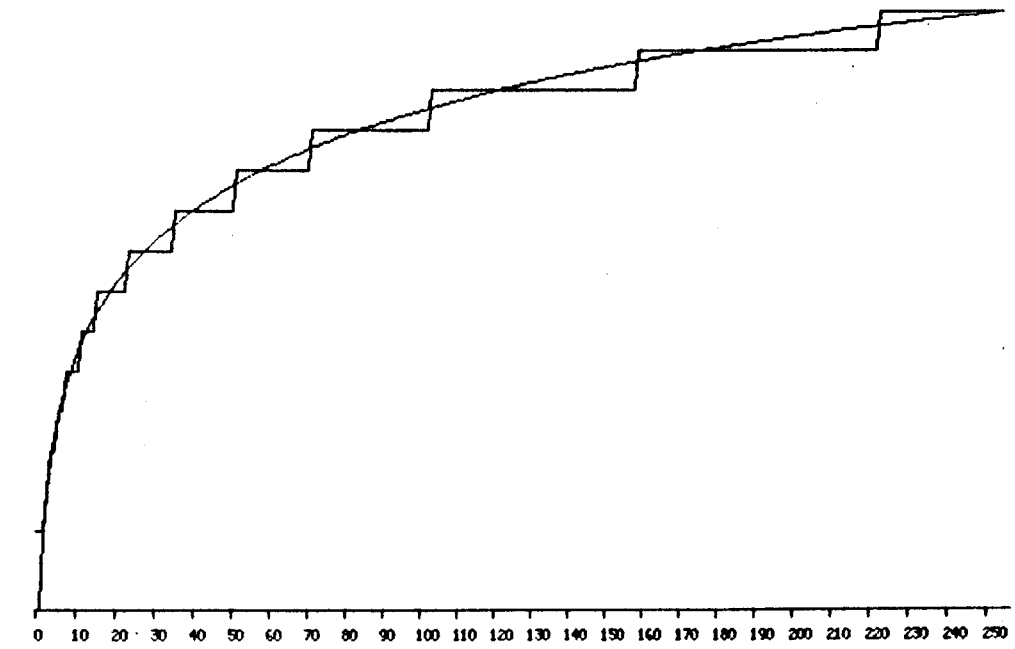


Figure 7: The LOG PLA. *(Top) Table used by LOG. The value produced by the PLA is given by the output number associated with the largest number in the* IN *column not greater than the input. (Bottom) A comparison of the results produced by LOG and the logarithm function. The two plots are normalized to give the same value when the input is 255. The standard error of the LOG PLA over the input range is 0.13.*

IN	OUT
0	2
1	3
2	4
-1	1
-2	0

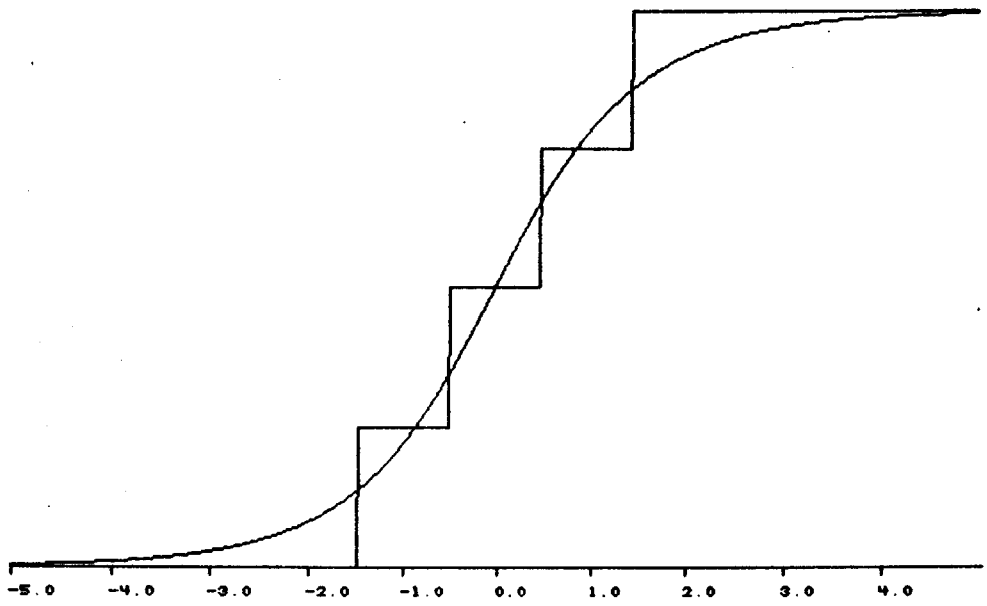


Figure 8: The ATN PLA. *(Top) Table used by ATN to compute two bits of the arctangent. Inputs less than -2 are given an output of 0, inputs greater than 2 are given an output of 4. (Bottom) Comparison of the results produced by the ATN PLA and the "arctangent of log" function in the first quadrant. The two plots are normalized to give the same value when the input is 0. The standard error of the entire arctangent calculation is 0.644*

IN	OUT	IN	OUT
44	1	104	3
48	1	112	3
52	1	120	4
56	1	128	4
60	1	144	5
64	1	160	7
72	1	176	8
80	2	192	10
88	2	208	11
96	2	224	13
		240	15

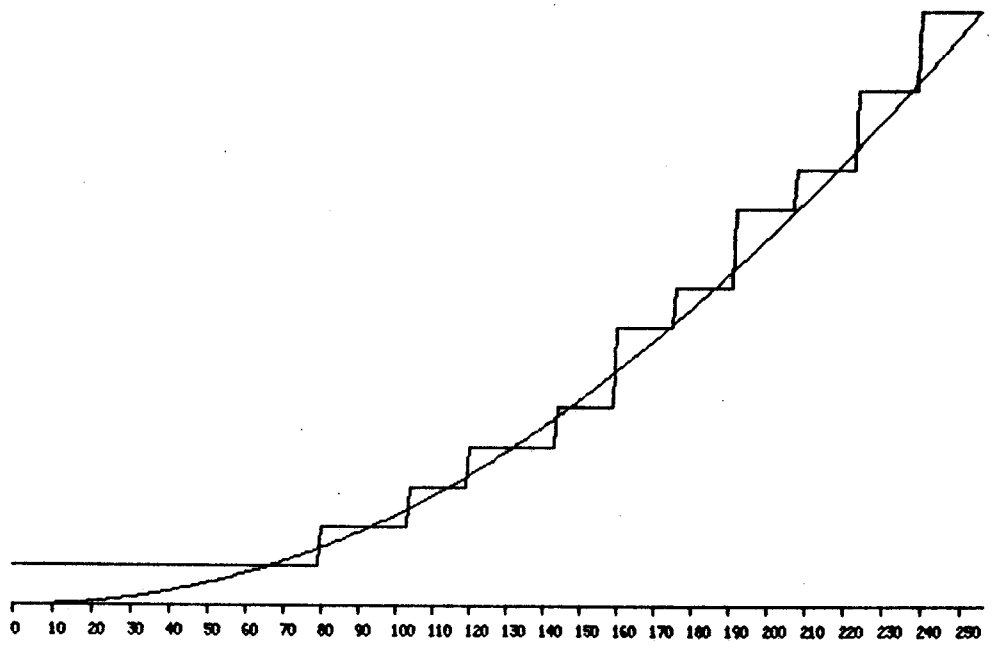


Figure 9: The X2 PLA. *(Top) Table used by X2. The value produced by the PLA is given by the element of the output column associated with the largest element of the* IN *column not greater than the input value. (Bottom) A comparison of the results produced by the X2 PLA and the "square" function. The two plots are normalized to give the same value for input 255. The standard error of the X2 PLA is 0.41.*

IN	OUT	IN	OUT
1	3	16	11
2	4	17	11
3	5	18	11
4	5	19	12
5	6	20	12
6	6	21	12
7	7	22	12
8	8	23	13
9	8	24	13
10	8	25	13
11	9	26	14
12	9	27	14
13	10	28	14
14	10	29	14
15	10	30	15
		31	15

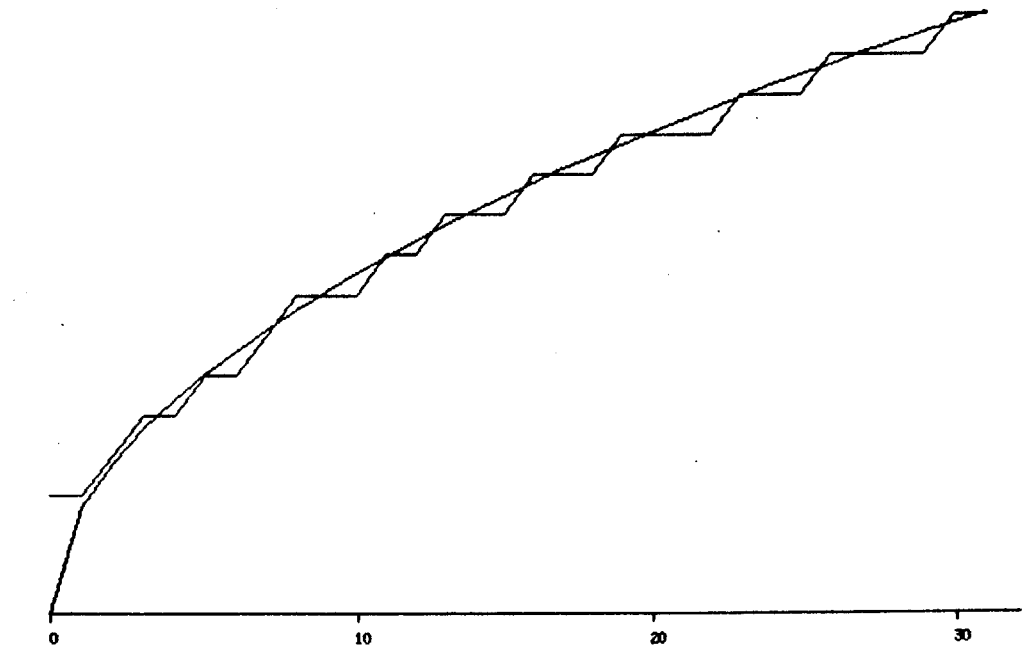


Figure 10: The SQT PLA. *(Top) Table used by SQT. The value produced by the PLA is given by the element of the output column associated with the largest element of the* IN *column not greater than the input value. (Bottom) A comparison of the results produced by the SQT PLA and the square root function. The two plots are normalized to give the same value for input 31. The standard error of the SQT PLA is 0.57.*

3.4 The Chip

A floor-plan of the DEL chip is shown in figure 11. It may be compared with the photograph of the chip in figure 1.

The input consists of two full 8-bit lines, taken by the groups of pads named LINE-1 and LINE-2 respectively, and the sign bit of the third line, taken by the pad SIGN-3.

The ZC pla performs the table lookup in figure 6. It is a finite-state machine, saving the values of the signs presented during the two previous clock cycles and using those, plus the values presented during the current cycle for the table lookup. The result, a single bit which is high if the central pixel contains a zero-crossing is sent into a shift register for 12 clock-cycles and output on the ZC-OUT pad.

The DIFF system consists of two 8-bit subtractors, and a shift register. The quantity **dy** is computed by subtracting the value of the LINE-1 input from that of LINE-2. **dx** is computed by taking the difference between the LINE-2 input and the value of the LINE-2 input from the previous clock cycle, which is stored in the shift register.

The two SM systems convert the differences into sign magnitude form. Each consists of a carry-chain, together with logic to detect the sign of the input. If the input is positive, it is passed through unchanged. Otherwise, it is complemented and incremented (the 2s complement operation). In both cases the sign and magnitude are output.

The data are now placed on the P-BUS, a solution to a rather thorny routing problem. Data enters the P-BUS and is sent to the LOG PLAs, which perform the table lookup of figure 7, and the X2 PLAs which perform the table lookup of figure 9. The outputs of these PLAs are then placed back on the P-BUS. The P-BUS also carries the signs of the differences from the SM systems to the GEN table to compute the rotation of the direction angle; it also carries the clock signals.

The direction calculation continues with the QUO subtractor, which sends its answer to the ATN table (figure 8). The sign values from the P-BUS are stored in a shift register so that the GEN signal will be ready when the output is available from the

ATN PLA. The results of these units are added in the ROT adder and the sum is sent to the 4 ATN pads.

The magnitude calculation uses the SMSQ adder to sum the results of the X2 tables. The result is sent through the SQT PLA whose table is shown in figure 10. The result is then saved in a shift register for 2 clock cycles so that it will appear on the 4 RSS pads during the same cycle as the corresponding ATN outputs.

A single timing regimen is obeyed by the entire chip, as well as its subsystems. Each functional object: the DEL chip itself, the adders, the PLAs; considers its input valid during PHI-1, and produces, and latches its result upon PHI-2 so that it will be valid for the next system during the following PHI-1 phase.

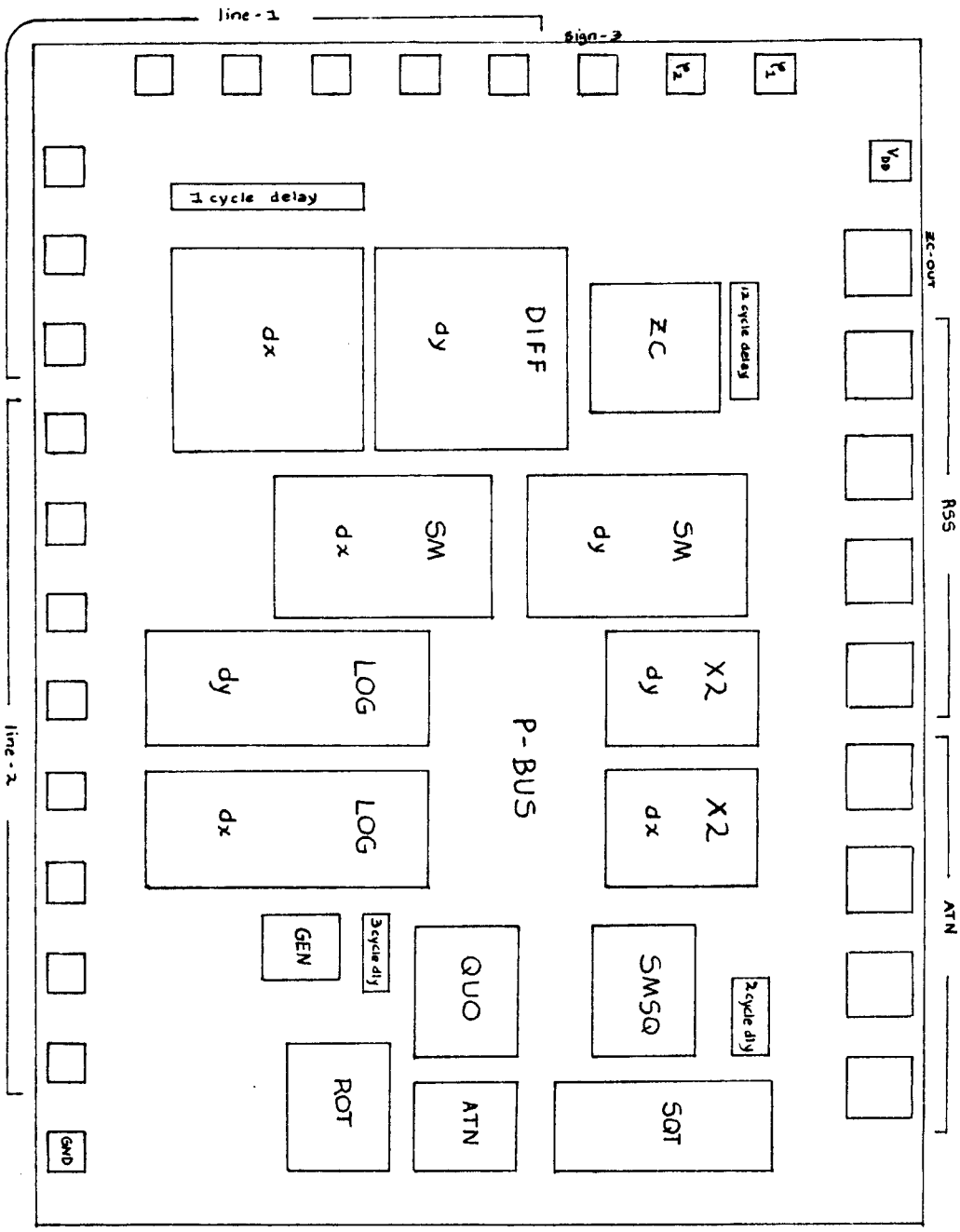


Figure 11: DEL floor plan.

3.5 The Pieces

DEL is constructed entirely from three kinds of modules, each slightly modified for the specific role it is to play. The modules are: PLAs, carry-chains and shift registers. This approach is much easier than one which would have required many different kinds of parts, and is also more reliable since a smaller number of basic building blocks must be constructed and tested.

The shift registers were the very simple shift registers described in [Mead & Conway, 1980]. A drawing of a shift-register cell is shown in figure 14. As mentioned in the previous section these are used at several points in DEL to hold data for several clock cycles. They are "noops" in the pipeline to make different length pipes merge at the correct time.

The PLAs were used for all table lookup operations as well as to implement the finite-state machine zero-crossing detector. The PLAs were produced by the PLA generator Jack Holloway wrote for the SCHEME-79 chip [Sussman, *et al*, 1979], and I modified.

The carry-chains are based on a Manchester Carry chain [Kilburn, *et al*, 1960] with a precharged carry line. A schematic diagram of a bit of the carry chain is shown in figure 12. This particular circuit is one bit of an adder. The carry-propagate (P) and carry-kill (K) signals are developed from the inputs during PHI-1 while the carry line (C) is being charged. During PHI-2 the carry is actually propagated and the output is developed from the P and Cin lines. The P and K signals are developed with full inverter logic during the clock cycle before the carry occurs. Similarly, the output bit is developed from the Cin and P signals during the clock signal after the carry occurs. Thus an entire operation of the carry chain takes 3 clock cycles. However each clock cycle may thus be relatively fast and, since the chip is intended to work in a pipelined manner, a fast clock period is the major speed constraint.

The carry chain in the SM subsystem is modified slightly from that discussed above. The top bit of the input data to the system is used to select one of two drivers that will feed into the rest of the bits during the PHI-2 clock cycle. The rest of the carry chain performs as a subtractor each cycle, but if the sign of the input is positive, the input is simply passed through unchanged, otherwise a complementation is performed. A signal developed from the sign bit makes the selection.

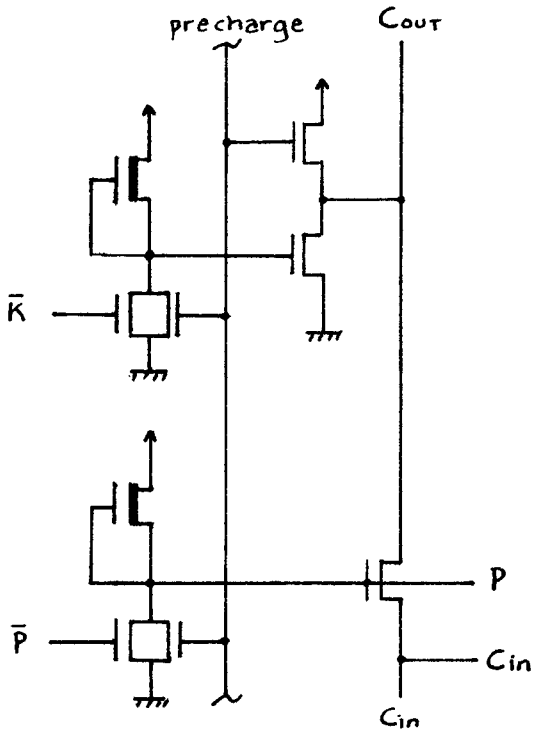


Figure 12: Carry-Chain Circuit. *The "kill" and "propagate" signals are developed from the input during the phi-1 clock phase, during which the precharge signal is high. Note that the precharge signal disables both the kill and the propagate signals. During phi-2 the* K *and* P *signals are complemented and fed into the carry chain. The output bit is developed from the* Cin *and* P *signals during the next clock cycle.*

3.6 Performance

The DEL chip was fabricated as part of the MPC580 multi-project chip set by XEROX PARC. I received two chips for testing. The chips were tested using the test facilities designed by Jon Taft [Taft, 1981].

Figure 13 presents the result of applying the zero-crossing detector in DEL to an interesting image. It may be compared with figure 1.

Testing of the gradient portions of the DEL chip was hampered by the fact that the processing of the chips was bad. Although both chips approximated the correct behavior in both the RSS and ATN outputs, neither had completely optimal performance and the two chips performed differently. However they work well

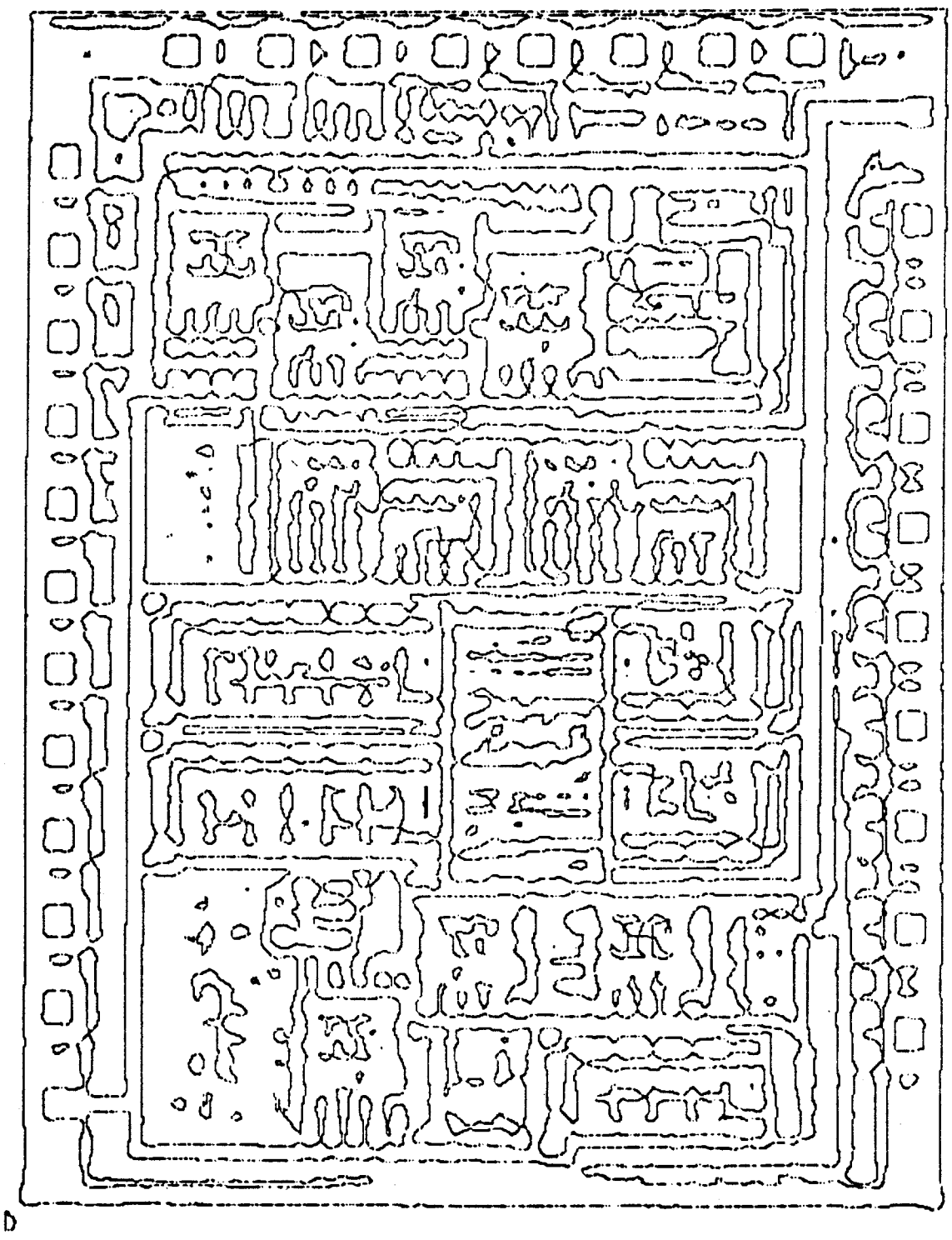


Figure 13: Silicon Self-Perception. *The image in figure 1 was convolved with a mask whose central panel width was 12 pixels. The result was then presented to the DEL chip.*

enough that they are both probably useful as they are for vision applications that need only the sign of the zero-crossing and a very rough measure of its slope.

The tester had the ability to clock the chip at various speeds. By increasing the clock rate while keeping the input constant, I was able to determine that the zero-crossing portion of the chip worked at 2 megahertz and failed at 4 megahertz. The gradient measurements seemed accurate at 1 megahertz and failed at 2 megahertz. These results satisfy the performance goals and indicate that the circuit could be used in a real time system.

3.7 Discussion

Perhaps the most novel feature of the DEL chip is the reliance on the PLA/Carry-chain architecture. By limiting most of the design to variations on these two building blocks, I was able to design more quickly and reliably than if a large number of special subsystems were used. The architecture seems suitable for any situation in visual processing where is is necessary to compute function which depends on the values of several adjacent pixels.

All of the table lookup PLAs were constructed by making the outputs depend on only the 3 or 4 most significant bits of the input. It turned out that this method was successful for my needs since it resulted in PLAs that were small enough to be practical, and furthermore the system as a whole was accurate to the 4-bit outputs produced. If more accuracy is required for other applications, a priority encoder could be used together with a table lookup. Such a system was actually designed for the DEL chip but was found not to be necessary for the accuracy desired.

The DEL chip essentially performs three calculations at the same time -- the zero-crossing detection, gradient direction, and gradient magnitude. This arrangement requires that the operations be done in parallel. Chips that must simply compute one value at each point -- for example: the convolution calculation -- could use serial operators at higher clock frequencies.

As I mentioned at the beginning of this section, my goals in this work were twofold: gaining experience at IC design, and producing a useful chip and thereby demonstrating that such hardware implementations of AI algorithms were possible. I

certainly have gained much experience in design. The most important lesson learned about IC design is that human beings don't want to do it. More will be said in the next chapter about support computers must give to the IC designer.

This project has demonstrated the feasibility of VLSI implementations of AI algorithms. Although I have not tested a perfect chip yet, the ones that I have received are useful as they are. I intend to design a slightly modified version of the chip soon. The new model would use faster PLAs and probably priority encoders to enable several bits more accuracy.

3.8 Criticisms

There are several valid criticisms that can be made to the approach presented here. They range from specific implementation details to the desirability of the implementation technology.

Although I optimized several cells for speed, the slowest operators in DEL, and thus the ones that most limits its speed, are the PLAs. Since the AND plane of the PLA is driven by inverters, its delay could have been reduced by using better drivers. I used the PLA generator that was available, however. The choice here, like that in many cases in the project, was for ease of design.

Computing the gradient by first differences is certainly not the most accurate choice. Other operators could have been implemented, perhaps with basically the same architecture. The approach was chosen because it was the easiest to implement. The first differences approximation to the gradient is more justified for this project than it might otherwise be for two reasons, both having to do with the fact that it is intended to be used to implement portions of the Marr theories: First, the theories require only a small amount -- a few bits -- of direction and contrast information. In fact, the theories really only require rough approximations to the gradient anyway, since they are intended to account for biological information processing. Certainly no biological system could rely on having perfectly accurate numerical calculations available. Second, the chip is designed to operate on an image that has been smoothed significantly by a gaussian convolution so that the first difference approximation to the gradient is more accurate than it would be in the more general case.

The DEL chip really performs two separate operations -- zero-crossing detection and gradient calculation. The zero-crossing detection is a very simple operation and could be performed by a simple comparator when the convolution is done. The gradient, on the other hand, could be computed by using a ROM as a table and looking up the result directly from the results of the first differences -- which could be computed with a pair of subtractors and a shift register. The ROM approach is also more flexible because any function of the first differences could be stored in the ROM. The general criticism is that by using VLSI, I have created an inflexible system that, arguably, doesn't even do the right thing.

This criticism is correct and more discussion of the relative merits of VLSI versus discrete implementations will be presented in the next chapter. Recall, however, that functionality was only one goal of the project.

Finally, it would seem that the vision machine architecture is rather limited. The only reason that the approach of having a number of operators on a serial stream can work, is that "later" computations don't communicate much with "earlier" ones. The hope is that enough can be done with this scheme so that later, more communication intensive processes, can work more leisurely on data that has been significantly reduced and summarized by the simple early processors.

Many AI algorithms, including some in vision, make use of extensive communication between processes. The stereo theory [Marr & Poggio, 1979] is "cooperative" and several other vision programs [for example: Ikeuchi & Horn 1981, Waltz, 1975] make use of constraints which must propagate information between operators at the "same" level of the computation. Development of the stereo theory and other vision work suggests that information flows "downward" at some points in the early processing of visual information.

These processes could not be implemented in the vision machine architecture. They seem to require a large number of communicating processing elements. Certainly a major area of research will be the study of possible and practical architectures for these processes. Study must be given to both the area of designing the operators, and abstractly describing what they are supposed to do.

4. VLSI DESIGN

If hardware is to have an impact on progress of AI, the design process must be as painless as possible. Most importantly, it must free the designer from details with which he is unconcerned. During my work designing DEL, for example, I had to consider details ranging from the transient behavior of nodes during inter-clock periods, to the best way to compute an arctangent. While it isn't clear which details the AI worker wants to grovel in, certainly they are more like the latter than the former. In this chapter I will discuss some issues relating to the design of a hardware implementation of an algorithm.

4.1 Choice of an Implementation Technology

In the previous chapter I described the implementation of a particular operator in a specific technology. The choice of a technology in which to implement a given operator depends on several factors, some intrinsic to the technology, and others that depend on the operator.

The most important choice to be made is that between the monolithic approach (LSI) and the discrete component approach. The primary advantage of the latter is that it is flexible and modifiable. The circuit may be debugged and altered if the original design is faulty or if the goals of the implementation change after the design is complete. An LSI design is, literally, "etched in stone" and alterations of existing chips are impossible. The discrete approach is also the most familiar. Parts, supplies and experienced designers are available. This factor also affects the cost of such an implementation.

LSI has the size advantage. An operator that would require many packages of TTL parts could fit in one LSI package. (This comparison isn't entirely fair -- the typical LSI chip needs several discrete components for support when it is used.) Power and speed considerations between the discrete and monolithic approaches aren't obvious yet but the trend seems to favor LSI. The monolithic approach also avoids the problems associated with the packaging of discrete components -- physical damage to chips, deterioration of bonding leads, etc. The more that can be done on a single chip, the less serious this problem. Also, LSI is conceptually easier to understand. The simple models of transistors as switches and resistors is adequate

for most of the design process. And since the designer is building a whole system, he doesn't have to cope with time multiplexing pins, or configuring an over-general TTL device to his specifications.

Some general conclusions: If size is at a premium, LSI is certainly the way to go. If the algorithm being implemented is not well understood and could be subject to change during or after the design process, a discrete component approach is indicated. If the algorithm is well understood and many copies of the hardware are planned to be build, LSI has advantages. If the algorithm is simple, and experienced designers and plenty of parts are available, components probably should be used. I think that while the increasingly complex candidate algorithms for hardware implementation will eventually swing the argument in the favor of LSI, this certainly is not true yet. The biggest obstacle in the way of the use of LSI is the difficulty of design.

4.2 The Design of DEL

In this section I present a chronology of the design process of the DEL chip. I feel that this should be instructive to those contemplating similar projects, and those seeking to locate the steps in the design process where automatic support is most critical.

During the fall term of 1979, I constructed a chip for the design project in Jon Allen's LSI design course. This project was a very simple zero-crossing detector chip. It had a fatal design bug and didn't work. (A wire was extended past the point it was meant to terminate and shorted a transistor.)

I began thinking about the possibility of the DEL chip during February 1980 when I became aware of a multi-project chip set being arranged by XEROX PARC for May of that year. During the later parts of February and the early weeks of March I had decided on an algorithm to implement. It seemed clear to me at that point that I would be relying on large PLAs for most of my calculations. I therefore wrote a program that simplified PLA programs to minimize the area of the resultant PLA. As it turned out, the design of the programming I ultimately used for the PLAs was such that this wasn't necessary.

By the 11th of April I had the basic architecture (as shown in figure 5) worked out. The programming for the PLAs was computed next. The most important part of this process was my understanding that the input planes for the PLAs could all use the priority encoding idea. All that remained was to encode the values of the functions being computed by the PLAs at each of the input points.

The deadline for the project chip was May 30th. The next weeks were spent doing the actual design. The modifications to the PLA program, and the design of the carry chain took about a week. Two weeks were spent putting together the major subsystems, and three were spent wiring them together. Around the 16th of May, I had completed a preliminary design.

The design of the chip was done completely textually, specifying the geometry with the layout language written by Gerald Sussman and Jack Holloway for the Scheme-79 chip. The language, a predecessor of that described in [Batali & Hartheimer, 1980] is a LISP-based language, in which the design is created by defining procedures that constructs a hierarchically ordered data-base. Different representations of the design may be extracted from the data-base, including the information to update the design if certain portions of it are altered, as well as the actual CIF file used to construct the chip. A graphics interface to the data-base was written for the LISP machine by Neil Mayle and I. This made it possible to view the results of the textual description. Interspersed with the actual design of the chip was time spent implementing features or fixing bugs of the design system.

An example of the language that I used to design DEL is shown in figure 14.

At this point the design was ready to be debugged. I used the software designed by Clark Baker and Chris Terman [Baker, 1980; Terman, 1981] for deign rule checking, node-extraction and simulation. Several design rule violations were found. The textual design language made it easy to modify the offending cells. The chip was then node-extracted and simulated and mostly worked the first time through. The most major bug located at this point was in the table that determined the rotation of the arctangent depending on the signs of the differences. This PLA was reprogrammed. The day before the deadline, the chip tested correctly and was submitted for processing.

While the construction of a chip of this complexity in roughly three

```
1       (DEFLAYOUT SHIFT-REG ()
2        (SET-THE 'ORIGIN (PT 0 0))
3        (SET-THE 'H-PITCH 19)
4        (SET-THE 'V-PITCH 19)
5        (SET-THE 'GND-HT 2)
6        (SET-THE 'VDD-HT 21)
7        (SET-THE 'INPUT (PT 4 13))
8        (SET-THE 'SLICE 14)
9        (SET-THE 'CIN 1)
10       (SET-THE 'COUT 20)
11       (SET-THE 'PU-PT (PT (THE SLICE) 1))
12       (SET-THE 'PHI-SLICE (THE CIN))

13       (SET-THE 'GND (WIRE (METAL 4) (PT 0 (THE GND-HT)) (X (THE H-PITCH))))
14       (SET-THE 'VDD (WIRE (METAL 4) (PT 0 (THE VDD-HT)) (X (THE H-PITCH))))
15       (SET-THE 'PHI1 (WIRE POLY (PT (THE CIN) 0) (Y (+ (THE V-PITCH) 4))))

16       (CALL 'BUTTING-CONTACT NIL 'ROT '(0 -1) 'TRANS (PT 6 13))
17       (CALL 'PULLUP '(6) '(TRANS (PT (THE SLICE) 11)))
18       (CONTACT S DIFF (PT (THE SLICE) (THE VDD-HT)))
19       (WIRE DIFF (PT (THE SLICE) (- (THE VDD-HT) 2)) (-Y 2))
20       (WIRE POLY (PT 7 13) (-Y 7) (+X 10))
21       (WIRE (DIFF 6) (PT 12 8) (-Y 4))
22       (CONTACT H DIFF (PT 12 (THE GND-HT))))
```

Figure 14: Example of layout language. *(Above) Textual specification of a cell. The cell will be one half-cycle of a shift-register. Line 1 indicates the definition of a kind of layout object named "shift-reg". This particular cell takes no arguments but in general arguments may be passed when the cell is instantiated, at which time the code constituting the body of the definition is evaluated. Lines 2-12 name some parameters and important points in the cell that will be useful while constructing the cell and later, when the cell is used. Lines 13-15 make wires and give them names. Lines 16 and 17 create parts of the cell -- a butting contact and a pullup -- by calling previously defined cells with arguments, and specify where they should be placed and what orientation they should have. Lines 18 and 19 connect the top of the pullup to the VDD wire. Lines 20 and 21 create the pulldown transistor and line 22 connects it to ground. The cell created by this definition is shown on the next page.*

---

graduate-student-months is something of an accomplishment, there are several aspects of the design process I used that could be improved. The most obvious is the lack of graphical feedback that was available. Although I described the chip textually, the only way to be sure that the description was correct was to see what it looked like. The graphics system that I used was rather low resolution and slow.

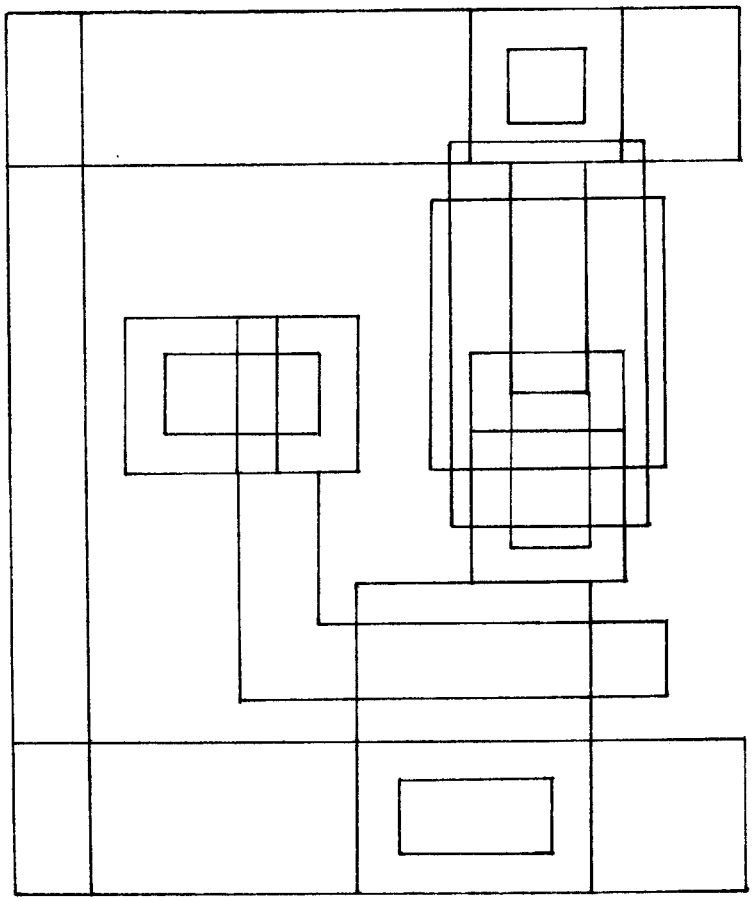


Figure 14 (continued): An instantiated SHIFT-REG cell. *See previous page for layout language code and discussion.*

Actually, although textual representation of the design is desirable to make cell descriptions that are parameterizable and flexible, it is often very difficult to make a textual description of a command that could easily be implemented by pointing at a particular position on a picture of the design. The Icarus [Fairbairn & Rowson, 1978] and Daedalus [Shrobe, 1981] design systems make it possible to design completely graphically, but then lose the modifiability of the resultant cells.

The single most painful part of the design process was the wiring. This phase

required that all the parts of the chip be viewed at once, to be sure the wires went where they were supposed to go, and taxed the graphics display system to the limit. Clearly reliable automatic routers are needed.

Finally, although the design verification tools I used were invaluable, it was inconvenient to have to run them only when the design was complete. A more useful approach might involve verification and simulation throughout the design process. One would describe the layout of the project, along with an electrical and functional description. These descriptions could then be compared with those determined by the verification tools and the designer notified of discrepancies.

4.3 Design Tools

The primary disadvantage of LSI for implementing algorithms is the inflexibility of the design. It is necessary that the design be correct before fabrication -- especially so if the cost of fabrication is high, whether measured in time or dollars. The advantages of discrete implementations would fade if satisfactory design aids for LSI can be developed.

If, as I suggested was desirable in chapter 2, many AI algorithms are to be implemented in hardware, it must be possible that the researcher who is to use a device be able to build it. However the researcher who is interested in a hardware implementation of his favorite algorithm doesn't want to learn electronics. Details removed from the actual specification of the algorithm must be automatic -- creating chips must look as much like programming as possible. The designer must be able to concentrate on the "algorithmic" aspects of the design, rather than the details of the particular fabrication technology.

The design system must be able to do more than just maintain a representation of a design. It must be able to do much of the detail work of the design process. Ideally the system would be like a compiler -- taking an algorithm expressed in a "high-level" language and producing a finished chip. The implementation may not be the fastest, smallest, or most efficient, but as in a compiler, the important thing is the ease with which it allows a large improvement in productivity on the part of the designer.

A somewhat less ambitious goal, but one whose result would still be very useful, is to provide the designer with a large number of tools at each point of the design process. Each tool would be optimized to do a particular job well -- routing, compacting a cell, node extracting, making plas, etc -- and it is up to the designer to decide when, where and how to use the tools.

An important part of this process is that the designer must be able to describe his design at several levels of abstraction. He may view the pieces of his design as "subroutines" for working out the algorithm, as modules to be wired together while he is instructing the routing program, and as design projects in their own right when the time comes to actually determine the details of the parts. It is important that support be given to these multiple representations. The standard layout languages usually just support the "rectangle" description of the design that is really most useful to the final fabricator of the design. The designer wants to think about rectangles as little as possible -- especially if the algorithms he is implementing are difficult, as they increasingly are. for example, the designer would rather spend his time considering details similar to those presented in figure 5 of this document, rather than those presented in figure 14.

Ideally, the design system would be able to insure that the different descriptions were consistent. For example the circuit description of a subsystem as described by the designer should match that determined by a node extraction program. I call this style of design "Incremental Verification." Such a design system allows the designer multiple representations of his work, and continually compares them with each other, notifying the him if inconsistencies arise.

The next step is to let programs actually fill in the details of the design. An example of this already widely used is a PLA generator. The programming of the PLA is typically described as Boolean equations, or perhaps as microcode, and the generator produces a layout from this specification. There are two important things about this process. First, the designer describes the PLA at the level he wishes to use it -- as a "function block" or a "micro-controller" rather than as a set of rectangles; second, once the PLA generator is complete and debugged, verifiers are unnecessary. Ultimately a design system would provide many analogous tools. The designer would be left with the job of specifying his algorithms and deciding which tools to use. Eventually, perhaps, "smart" programs could take the designer out of

REFERENCES

Baker, C. (1980) "Artwork Analysis Tools for VLSI Circuits," *MIT LCS Technical Report 234.*

Batali, J. & Hartheimer, A. (1980) "The Design Procedure Language Manual," *MIT AI Lab. Memo 598.*

Batali, J. & Ullman, S. (1979) "Motion Detection and Analysis," *ARPA Image Understanding Workshop,* L. Baumann (ed). Inc., Arlington, Virginia.

Doyle, J. (1980) "A Model for Deliberation, Action and Introspection," *MIT AI Lab. Technical Report 581.*

Dreyfus, H. (1979) *What Computers Can't Do,* Harper and Row, New York, 2nd Edition.

Fahlman, S. (1979) *NETL: A System for Representing and Using Real-World Knowledge,* The MIT Press, Cambridge, Massachusetts.

Fairbairn, D. & Rowson, J. (1978) "ICARUS: An Interactive Integrated Circuit Layout Program," *Proc. of the 15th Annual Design Automation Conf. IEEE.*

Fennema, C. & Thompson, W. (1979) "Velocity Determination in Scenes Containing Several Moving Objects," *Computer Graphics and Image Processing* **9,** 301-315.

Grimson, E. (1980a) "A Computer Implementation of a Theory of Human Stereo Vision," *MIT AI Lab, Memo 565.*

Grimson, E. (1980b) "Computing Shape Using a Theory of Human Stereo Vision," PhD Thesis, MIT Department of Mathematics.

Holloway, J., Steele, G., Sussman, G., & Bell, A. (1980) "The SCHEME-79 Chip," *MIT AI Lab. Memo 559.*

Ikeuchi, K & Horn, B. (1981) "Numerical Shape from Shading and Occluding Boundaries," *Artificial Intelligence* **(16).**

Horn, B. (1973) "On Lightness," *MIT AI Lab. Memo 295.*

Horn, B. & Schunk, B. (1980) "Determining Optical Flow," *MIT AI Lab. Memo 572.*

Kilburn, T., Edwards, D., Aspinall, D. (1960) "A Parallel Arithmetic Unit Using a Saturated Transistor Fast-Carry Circuit," *Proc. IEE* Pt. B. **Vol.** 107 November.

Logan, B. (1977) "Information in the zero-crossings of bandpass signals," *Bell System Technical Journal* **(56).**

Marr, D. (1976) "Early Processing of Visual Information," *Phil. Trans. R. Soc. Lond.* B 275 **(942).**

Marr, D. (1981) "Artificial Intelligence -- A Personal View," in *Mind Design* J. Haugland (ed) Bradford Books.

Marr, D. & Hildreth, E. (1979) "Theory of Edge Detection," *MIT AI Lab. Memo 518.*

Marr, D. & Poggio, T. (1977) "A Theory of Human Stereo Vision," *MIT AI Lab. Memo 451.*

Marr, D. & Ullman, S. (1979) "Directional Selectivity and its Use in Early Visual Processing," *MIT AI Lab. Memo 524.*

Mead, C. & Conway, L. (1980) *Introduction to VLSI Systems,* Addison-Wesley Publishing Co., Reading, Mass.

Miller, G., Galanter, E., Pibram, K. (1960) *Plans and the Structure of Behavior* Holt, Rinehart & Winston, New York.

Nishihara, K. & Larson, N. (1981) "Toward a Real-Time Implementation of the Marr-Poggio Stereo Matcher," *Proceedings of the ARPA Image Understanding Workshop* L. Baumann (ed).

Nudd, G., Nygaard, P., Fouse, S., Nussmeir, T. (1979) "Development of Custom-Designed Integrated Circuits for Image Understanding," *Proceedings of the ARPA Image Understanding Workshop* L. Baumann (ed).

Rivest, R. (1980) "A Description of a Single-Chip Implementation of the RSA Cipher," *Lambda* 1**(3)** Fall Quarter, 1980.

Shrobe, H. (1981) "The Daedalus Integrated Circuit Design Environment," *MIT AI Lab. Memo,* forthcoming.

Sussman, G., Holloway, J. & Knight, T. (1979) "Computer Aided Evolutionary Design for Digital Integrated Systems," *MIT AI Lab. Memo 526.*

Taft, J. (1981) "CHESTER the Tester," MIT AI Lab., forthcoming. forthcoming.

Terman (1981) *Simulation Tools for LSI Design,* MIT PhD Thesis, forthcoming.

Thompson, C. (1980) "A Complexity Theory for VLSI," PhD Thesis, CMU Department of Computer Science.

Ullman, S. (1979) *The Interpretation of Structure from Motion* The MIT Press, Cambridge, Massachusetts.

Ullman, S. (1980) "Interfacing the One-Dimensional Scanning of an Image With the Applications of Two-Dimensional Operators," *MIT AI Lab. Memo 591.*

Waltz, D. (1975) "Understanding Line Drawing of Scenes with Shadows," in *The Psychology of Computer Vision* P. Winston (ed), McGraw-Hill Book Company.

This empty page was substituted for a blank page in the original document.

CS-TR Scanning Project
Document Control Form

Date : 11 / 9 / 95

Report # AIM-869

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 56 (62 IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☒ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:

☒ DOD Form (2) ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number): _____

Photographs/Tonal Material (by page number): UN#'ED IMAGE 3, PAGE#'ED 11

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-56) UN#'ED TITLE, BLANK, ILLASTRATION, BLANK,	
3-53, UN#BLANK	
(57-62) SCANCONTROL, DOD(2), TRGT'S(3)	

Scanning Agent Signoff:
Date Received: 11/9/95 Date Scanned: 11/28/95 Date Returned: 11/30/95

Scanning Agent Signature: Michael W. Cook

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 869	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: AD-A162 536	
4. TITLE (and Subtitle): A Vision Chip	5. TYPE OF REPORT & PERIOD COVERED: A.I. Memo
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): John Batali	8. CONTRACT OR GRANT NUMBER(s): N0014-75-C-0643, N0014-80-C-0622, N0014-80-C-0505
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Artificial Inteligence Laboratory, 545 Technology Square, Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE: May 1981
	13. NUMBER OF PAGES: 53
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Office of Naval Research, Information Systems, Arlington, VA 22217	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

VLSI Design
Vision Hardware
Early Vision

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The time has come for workers in artificial intelligence to begin building hardware. The theories and algorithms being proposed in AI exceed thecapabilities of standard computers. Also, the understanding gained in the hardware implementation of a theory is probably not available any other way. The field of vision is one where progress awaits the speed that hardwareimplementations can provide. Some well understood and well justified algorithms for early visual procesing must be implemented in hardware for later visual processing to be studied. This paper desribes the design and hardware implementation

of a particular operator of visual processing. I constructed an NMOS VLSI circuit that computes the gradient and detects zero-crossings in a digital videoimage in real time. The algorithms employed by the chip, the design problems that had to be solved, and its capabilities and limitations are discussed.

The most difficult aspect of the construction of my vision chip was the attention that had to be paid to very low level detail. For hardware to be a useful tool for AI designing it must be as much like programming as possible. This paper concludes with some discussion of how such a goal can be met.

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**