1

MicroD

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

"VLSI TRIANGULATION PROCESSING FOR MACHINE VISION"

by

Todd D. Morris, B.A. Sc.

A thesis submitted to the Faculty
of Graduate Studies and
Research in partial fulfillment
of the requirements for the degree of

Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering
Department of Electronics
Faculty of Engineering
Carleton University

December 1987

© copyright

1987, Todd D. Morris

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-40638-0

The undersigned recommend to the Faculty of Graduate
Studies and Research the acceptance of the Thesis:

"VLSI Triangulation Processing for Machine Vision"

submitted by Todd D. Morris in partial fulfillment of the
requirements for the degree of Master of Engineering.

for Prof. M. Copeland

Chairman,

Department of Electronics

Prof. C.H. Chan

Thesis Supervisor

Carleton University

December, 1987

ABSTRACT

A VLSI solution has been used in the computation of triangulation formulae for a noncontact proximity sensor. The vision sensor can determine distance as well as surface orientation. Two VLSI solutions have been designed. The first method employs an architecture which is dedicated to the particular triangulation formulae for the above sensor. The second technique proposes a general purpose solution. Thus, a very fast digital signal processor has been designed which will solve triangulation formulae as well as perform other mathematical processing. Using the concept of multi-access global memory, this general purpose processor provides greater computational throughput than modern general purpose DSPs.

ACKNOWLEDGEMENTS

The author wishes to thank his fellow graduate students and the entire staff of the department of Electronics for creating an environment conducive to research. Particular thanks to my supervisor, Dr. C.H. Chan, to G. Bell for his CAD system's support, and to C.G. Cook for his CHIPSMITH development.

This research was made possible by the financial assistance of NSERC. The equipment and advisory support of the Canadian Microelectronics Corp. is also acknowledged.

TABLE OF CONTENTS

	Page
CHAPTER 1: INTRODUCTION	
1.0 Thesis Objective	1
1.1 Thesis Motivation	2
1.2 Background Information	3
1.2.1 Kanade and Sommer Work	3
1.2.2 University of Ottawa's Work	4
1.2.3 NRC's Vision Work	4
1.3 Thesis Format	5
CHAPTER 2: VISION SYSTEM DESIGN	
2.0 Introduction	6
2.1 System Overview	8
2.1.1 Sensor Characteristics	11
2.1.2 Sample and Hold Units and A/D Converters	13
2.1.3 Laser Diodes	14
2.1.4 VLSI Processing Unit	15
2.1.5 Host Computer Responsibilities	17
2.2 Optical Configuration	19
2.3 Triangulation Formulae	21
2.4 Applications	23
2.5 Summary	25

	Page
CHAPTER 3: DEDICATED TRIANGULATION FORMULAE	
3.0 Processing Requirements	26
3.1 Design Objectives and Constraints	26
3.2 Processor Architecture	29
3.2.1 System Pipeline	29
3.2.2 Control States	31
3.2.3 Input Stage	34
3.2.4 Divide/Multiply Stage	35
3.2.5 Output Stage	36
3.2.6 System Initialization	37
3.3 Design and Simulation	38
3.3.1 Divider	39
3.3.2 Multiplier	43
3.3.3 Adder Architecture	47
3.3.4 Subtractor Architecture	54
3.3.5 Square Root Architecture	63
3.3.6 Control Circuits	67
3.3.7 Registers	67
3.3.8 Bus Arbitration	71
3.3.9 Summary of Delays	71
3.4 Architecture Simulation Using SPLICE	71
3.5 Layout of the Dedicated TP	80
3.5.1 CAD Tools	80
3.5.2 Standard Cells	84

	Page
3.5.3 Simulation	84
3.5.3.1 Control Shifter Simulation	85
3.5.3.2 Register Simulation	85
3.5.3.3 Bus Arbitrator Simulation	85
3.5.4 System Layout	90
3.6 Test Results	90
3.7 Improvements	93
3.8 Applications	94
3.9 Summary	94

CHAPTER 4: GENERAL PURPOSE PROCESSOR

4.0 Introduction	95
4.1 Design Objectives and Constraints	96
4.2 Processor Architecture	97
4.2.1 Processing Units	97
4.2.2 Instruction Cache	102
4.2.2.1 Instruction Set	104
4.2.3 Data Memory, Busing, and Address Decoding	106
4.2.4 Input and Output Stage	109
4.3 Triangulation Formulae Performance	110
4.4 Comparison with Other Processors	111
4.5 Enhancements and Improvements	114
4.5.1 Program Memory	115
4.5.2 Data Memory	116

	Page
4.5.2.1 Requirements and Constraints	117
4.5.2.2 Memory Chip Architecture	118
4.5.2.3 Collision Detection	121
4.5.2.4 Other Circuitry	124
4.5.2.5 Memory Cell Design	128
4.5.2.6 Enhancements and Conclusions	136
4.6. Applications	138
4.7 Summary	139
 CHAPTER 5: CONCLUSION	
5.0 Summary	141
5.1 Recommendations	142
 REFERENCES	 143

LIST OF TABLES

Table	Page
3.1 Divider Comparison	40
3.2 Multiplier Comparison	45
3.3 Multiplier Simulation Results	49
3.4 Adder Simulation Results	53
3.5 Subtractor Simulation Results	62
3.6 Square Root Simulation Results	66
3.7 Summary of Component Delays	73
4.1 Comparison of GTPs	140

LIST OF FIGURES

Figure	Page
2.1 Overall Vision System	9
2.2 PSD Diagram	12
2.3 Optical Configuration	20
2.4 Optical Geometry	22
3.1 Dedicated TP Architecture	30
3.2 Shifter7 Architecture	33
3.3 Divider Architecture	42
3.4 Divider Simulation	44
3.5 Multiplier Architecture	46
3.6 Multiplier Simulation	48
3.7 Adder Architecture	50
3.8 8-bit Adder Simulation	51
3.9 16-bit Adder Simulation	52
3.10 SAS Architecture	56
3.11 TSAS Architecture	57
3.12 SAS Simulation	58
3.13 TSAS Simulation	59
3.14 Subtractor Architecture	60
3.15 Subtractor Simulation	61
3.16 Square Root Architecture	64
3.17 Square Root Simulation	65

Figure	Page
3.18 Shifter5 Simulation	68
3.19 Shifter7 Simulation	69
3.20 Register Simulation	70
3.21 Bus Arbitrator Simulation	72
3.22 SPLICE Architecture	74
3.23 SPLICE Control Unit Simulation	77
3.24 SPLICE Input Circuitry Simulation	78
3.25 SPLICE Internal Units Simulation	79
3.26 SPLICE Output Circuitry Simulation	81
3.27 Hg Shifter5 Simulation	86
3.28 Hg Shifter7 Simulation	87
3.29 Hg Register Simulation	88
3.30 Hg Bus Arbitrator Simulation	89
3.31 IC1 Layout	91
3.32 IC2 Layout	91
3.33 IC1 and IC2 Interconnectivity	92
4.1 GTP Architecture	98
4.2 Square/Square Root Architecture	101
4.3 Multi-Access RAM Architecture	119
4.4 Collision Detector	122
4.5 Collision Detector Simulation	123
4.6 ATD Circuit	125
4.7 SAT Circuit	125
4.8 ATD Simulation	126

Figure	Page
4.9 SAT Simulation	127
4.10 Decoder Circuit	129
4.11 RAM Cell Circuit	130
4.12 SPICE Pre-Layout Simulation	132
4.13 SPICE Input Waves	133
4.14 RAM Cell Layout	134
4.15 SPICE Post-Layout Simulation 1	135
4.16 SPICE Post-Layout Simulation 2	137

LIST OF APPENDICES

	Page
Appendix A : SPLICE Models	145
Appendix B : CHIPSMITH Model	154

GLOSSARY OF SYMBOLS AND ABBREVIATIONS

2-D	Two Dimensional
3-D	Three Dimensional
A/D	Analog to Digital
ATD	Address Transition Detector
car _i	Carry Input
car _o	Carry Output
CAS	Controlled Add/Subtract Cell
CASF	Controlled Add/Subtract Shift Cell
CG	Carry Generation
CLA	Carry Look Ahead
CMC	Canadian Microelectronics Corporation
CS	Chip Select
D	Delay
DMA	Direct Memory Access
DSP	Digital Signal Processor
GPTP	General Purpose Triangulation Processor
FA	Full Adder
IC	Integrated Circuit
IR	Instruction Register
NOP	No Operation
NRC	National Research Council
NT	Northern Telecom
PC	Program Counter

POR	Power On Reset
PSD	Position Sensitive Detector
RAM	Random Access Memory
S/H	Sample and Hold
SAS	Sign Adjust
SAT	Stable Address Transition
SG	Sum Generation
Sqroot	Square Root
TF	Triangulation Formulae
TP	Triangulation Processor
TSAS	Terminating Sign Adjust
WR	Write Signal

CHAPTER 1: INTRODUCTION

1.0 Thesis Objective

With advances in Integrated Circuit (IC) technology, complex single chip processors have become readily available. This has allowed high processing dependent applications such as artificial vision to become faster, less cumbersome, less costly, and thus more viable. Typically, depending on the number of units to be manufactured, such applications will either use a dedicated or a general purpose processor solution. Generally, the dedicated solution provides better performance at a greater cost.

This thesis examines a particular class of artificial vision. This class of vision, termed the "point method", requires triangulation formulae processing. A dedicated and general purpose solution has been developed to compute these formulae. These two solutions will be analyzed and compared on the basis of performance and economic feasibility. From the design and development of these two VLSI solutions, some novel techniques for parallel processor and multiprocessor architectures will be discussed.

The results and discussions presented herein reflect the conclusions drawn from design, simulation, and

testing, as well as current literature.

1.1 Thesis Motivation

The thesis motivation can be broken down into two main categories: vision system motivation and single chip general purpose multiprocessor motivation. The vision system itself can be used for advanced research in many image processing applications. Previously, image systems employing the point technique suffered from lack of speed. Therefore, there was a great motivation to increase the vision speed in order to establish the "point method" as a viable technique. A dedicated VLSI architecture was needed to improve the performance (i.e. throughput) of similar previously developed vision systems. This resulted in a bottleneck at the sampling sensor itself, and improvement of this sensor is beyond the scope of this thesis.

The main motivation behind a general purpose solution was to accomodate future sensor advancements which may result in a different set or class of triangulation formulae (TF). It was decided that the general purpose solution should provide comparable performance to the dedicated processor in order to remain competitive. Advanced techniques were required to achieve this which resulted in a single chip multiprocessor utilizing a global

memory configuration. By employing a general purpose solution and high speed techniques, many other digital signal processing (DSP) applications besides triangulation processing can be performed.

1.2 Background Information

1.2.1 Kanade and Sommer Work

In 1983 at the Robotics Institute at Carnegie-Mellon University, Kanade and Sommer developed an Optical Proximity Sensor for measuring surface position and orientation for robot manipulation. Their vision system employs the "point method" and is very similar in system configuration to the one proposed herein. The Carnegie-Mellon system used a microcomputer for triangulation processing and thus, relatively slow performance was obtained. Due to the implementation and testing of the system at Carnegie-Mellon and a detailed outline of the vision sensor in a paper by Kanade and Sommer [14], much of their design and test results could be assumed correct and research into improving their system could be examined.

1.2.2 University of Ottawa's Work

The University of Ottawa proceeded in further development of Kanade and Sommer's vision system. Additional circuitry was designed to improve the accuracy and reliability of the system. In 1985, the University of Ottawa examined the use of VLSI circuitry to improve the vision system performance. A set of ICs was designed and implemented. These ICs performed some initial processing of signals but still relied heavily on a microprocessor for solving the triangulation formulae. Although these ICs were not used in the solution proposed herein, some of the smaller building block configurations were deemed suitable and thus incorporated in the designs [26].

1.2.3 NRC's Vision Work

A similar vision system is presently being employed at NRC. The actual optical configuration and sensor in use is different, but the system returns the same type of data to the user. NRC has developed algorithms to manipulate this raw data to form very detailed three dimensional images. In fact, algorithms have been successfully implemented which allow a robot to grasp and distinguish overlapping parts. These algorithms could similarly be used with the Kanade and Sommer system as well as with the system proposed herein [27].

1.3 Thesis Format

The main objective of this thesis is to design a vision system employing the "point method" which utilizes VLSI processing circuitry. Chapter 2 will provide an overview of the vision system which expands on Kanade and Sommer's work. The sampling technique, optical configuration, and triangulation formulae (TF) requirements will also be examined. Chapter 3 will provide the full design cycle for the VLSI dedicated solution to the TF. This cycle will include design objectives, architecture trade-offs, simulation, and test results. Chapter 4 will examine the use of a general purpose solution to the TF. A design will be presented as well as a comparison with the dedicated processor and other commercially available ICs. Some advanced concepts for multiprocessor systems will be examined and further developed in the Improvements section for the general purpose processor. Conclusions and recommendations will follow in Chapter 5.

CHAPTER 2: VISION SYSTEM DESIGN

2.0 Introduction

There are many possible methods which can be employed in a vision system. Following a given set of criteria, the most suitable method can be determined. The objectives of a noncontact vision system providing a relatively high speed, object independence, low processing requirements, and low cost form difficult criteria to achieve. Due to the recent cost effectiveness of optical means and their noncontact nature, an optical vision system was deemed most appropriate. One of the most commonly used optical based vision systems employs a camera as the transducer. Typically, the camera has a 512 x 512 pixel resolution with 256 levels of grayness per pixel. Such a transducer can record an entire scene with a single sample. There are several drawbacks to this method which do not satisfy the previously discussed criteria. To store and process such images requires large amounts of memory and processing power. This in turn increases the cost. Another problem is how to use the processed data. The image is in two dimensions only which makes object recognition extremely difficult. More images (samples) could be taken from different perspectives to alleviate this problem but this in turn increases processing requirements and thus cost[4].

One can see that the camera method satisfies only some of the criteria and thus a more suitable method should be determined.

A more suitable technique is the "point method". This method samples individual points directly on an object's surface. The individual points are targeted using an optical device such as a laser or laser diode. A suitable transducer such as an optical position sensitive detector (PSD) can record the single point sample. Triangulation can then be used to return a cartesian spatial coordinate of the point on the object's surface. The speed of this method is mainly determined by the transducer due to the sequential nature of the sampling. For very high resolution requirements, the sequential sampling is a definite drawback in terms of speed. For most objects, very high resolution is not required and thus, speed is not a problem. Comparing this method with the criteria, the point method provides noncontact vision, object independence, low processing requirements (triangulation formulae), and cost mainly dependent upon the transducer. Another benefit of the point method is the inherent programmable resolution. In the camera method, even for a simple object, a whole scene must be recorded and processed. For simple objects using the point method, less samples can be taken. Therefore, due to the above.

properties of the point method as well as previous documented research in this area, the point method was chosen to be further developed.

2.1 System Overview

The basic system configuration is based upon the proximity sensor implemented by Kanade and Sommer in 1983 [14]. The proposed system contains 16 laser diodes, one two dimensional position sensitive detector (PSD), four sample and hold units with amplification, four analog to digital (A/D) converters, two custom VLSI processing ICs, which form the triangulation processor, and a host microcomputer. The overall system configuration can be seen in figure 2.1.

The laser diodes are arranged in a circular fashion around the PSD. When a single laser diode is enabled, the PSD will transduce the reflected signal from the object's surface. The PSD transmits four low intensity currents to the sample and hold units and then to the A/D converters. These four digital values are sent to the two custom ICs which form the triangulation processor. The processor receives the data and triangulates to obtain the spatial (x,y,z) coordinates of the point on the object's surface. This processed data can then be transmitted to the host

Figure 2.1 : Overall Vision System

microcomputer. The system configuration is identical to that of Kanade and Sommer except for the addition of ten more laser diodes and the triangulation processor. Kanade and Sommer were able to achieve 1KHz throughput on very simple plane surfaces. For complex shapes, this speed would not be acceptable since several thousand points may be required and much speed may be lost in moving the vision sensor around an object in order to obtain a new set of sixteen samples. Note that for a particular fixed vision system location, at most 16 points on a stationary object can be sampled. For more points, the vision system must be moved to a new vantage point. Much speed will be lost in moving the system and thus, the triangulation processor is required to increase the processing speed. A 10KHz (or better) system throughput of individual samples is desirable. Typically, the vision sensor will be mounted on a robot arm. Since most robot arm motion is controlled by stepper motors, high processing speed from the vision system will enable the robot arm to move continuously around an object. In other terms, 16 samples per robot arm step would be achieved without the arm slowing or stopping. These are all contributing factors which point to increased speed requirements and thus, the need for the triangulation processor.

2.1.1 Sensor Characteristics

The transducer which is the main determinant factor in the cost, speed, and accuracy of the vision system is a silicon two dimensional position sensitive detector (PSD). The PSD which is the Hamamatsu S1300 works on the following principle. The PSD is basically a PIN photodiode with uniform resistive sheets on both sides. Four electrodes are used to collect the photocurrents. When a spot of incident light strikes the PSD, electric charge directly proportional to the light energy or intensity is generated. This charge then drives through the resistive sheets to the electrodes to form the photocurrent. The photocurrent is inversely proportional to the incident light spot's distance from the electrode. The functional diagram of the PSD can be seen in figure 2.2. If an incident light spot arrives at (X_i, Y_i) , then the four associated photocurrents will be given by:

$$I_{x1} = I_o(L - X_i)/2L \quad I_{x2} = I_o(L + X_i)/2L$$

$$I_{y1} = I_o(L - Y_i)/2L \quad I_{y2} = I_o(L + Y_i)/2L$$

where $I_o = \text{Total photocurrent} = I_{x1} + I_{x2} = I_{y1} + I_{y2}$

and the sensitive array area has dimensions $2L$ by $2L$.

To recover the actual (X_i, Y_i) coordinate of the incident spot, simple mathematical formulae can be determined.

$$X_i/L = (I_{x2} - I_{x1}) / (I_{x2} + I_{x1})$$

$$Y_i/L = (I_{y2} - I_{y1}) / (I_{y2} + I_{y1})$$

Figure 2.2 : PSD Characteristics

The PSD intrinsically locates the intensity weighted center of light distribution [10]. Thus, the system automatically resolves malformed or nonfocused incident light spots.

As mentioned previously, the transducer must now be examined for its effect on the cost, speed, and accuracy of the vision system. The cost of the Hamamatsu S1300 is approximately \$1000 U.S. when purchasing a single unit. This will form much of the system cost. As will be mentioned later (Chapter 4), newer, cheaper sensors have been developed. The PSD has a rise time of 8 μ s and an accuracy ranging from 80 to 150 μ m. Kanade and Sommer produced accuracy of 0.07mm for their entire system [14]. The speed of the PSD means that even using a totally pipelined system, the maximum data rate is 1/8 MHz.

2.1.2 Sample and Hold Units and A/D Converters

Receiving the signals from the PSD are the sample and hold (S/H) units and the A/D converters. The purpose of the S/H units is to provide matched input impedance while amplifying and converting the PSD output currents to analog voltages. These analog voltages are then converted to the digital domain by the A/D converters. Eight bits of accuracy was used since this provides slightly greater accuracy than the PSD. The goal of 10KHz system throughput forced a 25KHz triangulation processor (TP) throughput.

This 25KHz data rate can be used as a starting basis for the performance requirements of the S/H units and the A/D converters. Given a pipelined system architecture, the data will have 40us to be processed by the PSD, S/H units, and the A/D converters. Halving this time to ensure reliability and performance results in 12us for the S/H units and A/D conversion. Since each of the four PSD currents has a separate S/H unit and A/D converter, this 12us restriction is a viable constraint.

2.1.3 Laser Diodes

The light sources to be chosen are dependent upon the desired range of the vision system. For a 4 to 10 cm range, focusing LEDs or laser diodes provide sufficient output power for the PSD to detect the light spot. LEDs were used by Kanade and Sommer (Litronix SFH400-3). More powerful laser diodes are available today. For a greater sensing or viewing distance, more powerful light sources are required. This can be seen more clearly by examining the power reaching the PSD. The incident power can be estimated by

$$\text{Incident Power (PSD)} = (A^2 P_s) / (4 L^2)$$

where A is the diameter of the lens placed in front of the PSD, P_s is the light source power, and L is the distance to the object.

It is evident that as the desired viewing distance increases, the input power to the PSD follows the inverse square law. The chosen PSD (Hamamatsu S1300) requires a minimum of 2uW to operate. Thus, using $L=1m$ and $A=1cm$

$$P_s(\min) = 4P_{psd}(\min) * L * L / (A * A)$$

$$= 80mW$$

80mW of light source power is not an easy task to achieve for a LED or a laser diode, particularly since this is an absolute minimum value. A poorly reflective object will require far more light source power. It should also be noted that focusing light sources are inherently required to provide a relatively small light spot on an object's surface.

The final decision in choosing the light source is the desired light frequency. The PSD is most receptive to the infrared band (approximately 900nm), and thus, infrared sources were chosen. This also allows easy filtering of background white light. In a highly industrialized setting, the vision sensor may be prone to infrared noise from surrounding machinery.

2.1.4 VLSI Processing Unit

The purpose of the triangulation processing unit is to relieve the total processing burden from the host microcomputer. In this way, both the triangulation

processor (TP) and the microcomputer can function in parallel. The TP must be capable of accepting the digital inputs from the A/D converters, processing the data, and then effectively communicating the results to the host microcomputer. Thus, a communication protocol must be established for the data acquisition and the data output between the appropriate units. There is a large amount of processing required at relatively high speed. The TP must first calculate the two dimensional (2-D) incident light spot coordinates on the PSD (formulae derived previously). These 2-D coordinates must then be used to triangulate into three dimensions (3-D). The exact triangulation formulae will be derived in a later section. The cartesian 3-D coordinates of the spot on the object's surface can then be communicated to the host microcomputer. Since a system throughput of 10KHz was desired, it was estimated that a 25KHz TP throughput was required for a safe design margin.

Due to the processing requirements (seen later in the triangulation formulae), the use of a microprocessor or similar device cannot provide sufficient speed. Currently, the TP was designed with an 8-bit accuracy, but in order to maintain flexibility for future advanced sensors, a lookup table approach was deemed not suitable. A bit-slice was not used due to its cumbersomeness. Thus, to provide adequate throughput as well as maintaining the same basic architecture despite future advances in sensor

technology, a custom VLSI circuit was chosen. The custom VLSI TP will be examined in much greater detail in chapters 3 and 4 [18].

2.1.5 Host Computer Responsibilities

The host microcomputer has several tasks to perform. The computer acts as the overall system controller. Thus, it controls the enabling and disabling of the light sources; the activation of the sample and hold units; the control of the TP; and some initial system power up procedures. The controller functions will take the form of an interrupt routine. The other function associated with the host computer will be to further process the 3-D coordinates received from the TP. The exact processing will depend upon the application. It may vary from plane fitting (Kanade and Sommer); to pick and place applications (NRC); and even to object recognition. These applications will not be examined in greater detail as they have already been developed.

By referring to figure 2.1, the controller (host microcomputer) tasks can be further analyzed. Upon system power up, the host computer downloads several system configuration constants to the TP. These constants will be derived later when determining the triangulation formulae. The constants are downloaded on the output bus through the

use of the Constload and SelConst signals. By not hardcoding these constants, the system becomes somewhat reconfigurable in terms of optical configuration. Once the system is running, an interrupt signal from the TP will be issued to the host computer. The computer will then enable the next light source while disabling the current one; gather the three cartesian coordinates from the TP's output bus through the use of the Select Output signal; and then issue a START signal to restart the system for the next sample.

The required speed of the host computer is determined by the application. High resolution requires many coordinates to be stored and further processed resulting in the need for a powerful host computer. Using a safe estimate of 20us throughput from the TP, the host computer should respond to an interrupt, gather data, control the light sources, and restart the system in this time. Considering that some of the control bits can share the same computer port, some of these tasks can operate in parallel. If the host computer has this speed capability, then the computer could appear transparent to the TP in terms of communication delay. If however, the host computer cannot complete its required tasks within the given time period, the entire system will have to wait. This is accomplished by halting the TP after it interrupts

the host computer. A fast host will immediately restart the TP, whereas a slower host will perform some of its necessary tasks before a restart is issued.

2.2 Optical Configuration

The optical configuration can be seen in figure 2.3. At most one light source is enabled at any time. When one of the light sources is enabled, the respective light energy is reflected off the object's surface, collected by the gathering lens, and detected by the PSD. After the appropriate sensor delay (8 μ s for the Hamamatsu S1300), the four PSD output currents are valid. After this set of four currents has passed the S/H units and the A/D converters, a different light source is enabled while the previous one is disabled. Thus, a new spot on the object's surface is sampled. Since 16 light sources are used, a set of 16 samples can be obtained for each location of the vision system. Of course, if the vision system is mounted on a robot arm or a similar device, the vision system can take 16 samples from many different perspectives or vantage points. Therefore, the resolution (number of samples) is totally programmable.

Figure 2.3 : Optical Configuration

2.3 Triangulation Formulae

Taking a cross section of the optical configuration, as shown in figure 2.4, the optical geometry can be examined. From figure 2.4, the following triangulation formulae can be derived:

$$\tan \theta = -y'/L = y/z$$

$$y = (-y'/L)*z \quad (1)$$

$$\text{Similarly, } x = (-x'/L)*z \quad (2)$$

$$\text{Sqrt}(x*x+y*y) = (z/L)\text{Sqrt}(x'*x'+y'*y')$$

$$\text{Let } S = \text{Sqrt}(x'*x'+y'*y') \quad (S1)$$

= radial distance to incident light on the PSD

S could be expressed trigonometrically as

$$S = x'/\sin[\arctan(x'/y')] \quad (S2)$$

$$\text{Sqrt}(x*x+y*y) = z*S/L$$

$$\tan \theta = (r - \text{Sqrt}(x*x+y*y))/(L+z)$$

$$z = L(r - L\tan \theta)/(S + L\tan \theta) \quad (3)$$

There are three constants required: L, Ltan θ , and L(r-Ltan θ). As was mentioned previously, these three constants are loaded at system initialization or power up. In this way, if the radius of the light sources is changed (r), or the angle of light projection is altered (θ), the system can easily be reconfigured with the new constants.

Since two possible formulae (S1 and S2) can be used in the computation of S, the most suitable must be determined. From examining the two formulae for S, it is evident that

Figure 2.4 : Optical Geometry

either a square root, an addition, and two multiplications, or two divisions and a lookup table reference must be performed. By noting that five other divisions are already required compared to two multiplications, formula S1 seems more appropriate. Later it will be shown that the multiplications, square root, and addition can actually take place in parallel with the other five divisions. This of course would not be possible if formula S2 were chosen. An added square root circuit or method is required, but the lookup table is eliminated.

2.4 Applications

There are many suitable applications for this vision system ranging from a research environment, to a hospital, and of course to an industrial plant. A similar vision system is already being employed at NRC for research purposes. Research into object recognition and manipulation is suitable subject matter. Typically, the vision sensor will be mounted on some sort of robot arm so that many samples can be taken of an object. This programmable resolution allows the vision system to be applied to a full complexity range of tasks.

There is some interest in using a vision system to aid disabled persons in a hospital environment. Such a vision

aid could be used to enhance already existing equipment such as automatic feeders. The vision system could easily detect the presence and location of food as well as an individual's mouth when mounted on the feeder's arm. There would be some noise problems associated with the infrared radiation from the hot food.

Most of NRC's research and such hospital aids as mentioned fall under the broad title of "pick and place" applications. Industry makes the most use of the pick and place operation in automated plants. This operation is used to move objects and orient them correctly. The main advantage of using a vision system in conjunction with a robot is to direct or control the robot in such a manner as to make the object's orientation non-critical. Thus, no matter how the object is positioned, the vision system can determine how the object should be grasped. NRC has developed algorithms to determine "the most suitable hold site" on an object. Using a similar vision system to the one proposed herein, NRC has been successful in grasping one object from a group of overlapping objects. Placing the object is also aided by the vision system. In order to find the correct placement location or desired orientation (if necessary), the vision system can be used.

Therefore, due to the lightweight nature of the system (allows its use with a robot arm), the vision system can

aid in many pick and place applications. The applications can range from helping the disabled to automated assembly lines.

2.5 Summary

The point method of machine vision seems to provide many benefits over other possible methods: such as, low cost, programmable resolution, low processing, relatively high speed, and object independence. Furthering the work of Kanade and Sommer, a high speed vision system employing the point method has been designed. In order to increase the speed, a custom VLSI triangulation processor solution has been deemed suitable. The actual triangulation formulae have been further developed to be useful in cartesian coordinates (Kanade and Sommer used polar coordinates). In conclusion, the entire vision system appears to satisfy many applications at a relatively low cost. Such applications range from research to industrial uses.

CHAPTER 3: DEDICATED TRIANGULATION PROCESSOR

3.0 Processing Requirements

Custom or semi-custom VLSI circuits can be designed to meet the required objectives as long as the chosen process is capable. The relative low cost of silicon makes these circuits attractive. Of course, the low cost only appears if a large number of identical ICs are being manufactured. For a low volume run, the engineering design cost added to the initial mask making cost will not make custom ICs very attractive. Due to the large number of applications for the vision system, a VLSI solution is probably economically feasible. As discussed previously (Chapter 2), the custom VLSI TP provides a better solution (functionally and economically) than off the shelf products. Since the need for a custom VLSI TP has been determined, the processing requirements and constraints must be established before the design of the TP architecture and circuitry.

3.1 Design Objectives and Constraints

There are several processing requirements and constraints which must be satisfied. These parameters are listed below:

- 1) The ability to process the previously derived

triangulation formulae.

2) The ability to communicate simply with the A/D converters and the host computer.

3) To work within the boundaries of a certain technology, silicon area, and pin count.

4) To provide the required speed and accuracy.

Each of these requirements will now be discussed in greater detail.

As can be seen from the triangulation formulae, purely mathematical operations are required. Thus, any ALU could perform the formulae. Other processing units could also be employed depending on speed considerations which will be discussed under requirement 4.

Simple communication must be established with the A/D converters and the host microcomputer. A separate suitably lengthed bus to each of these units would be best. With four A/D converters, this may not be feasible and thus, the A/Ds may have to time multiplex on one bus. Similarly, the three (x,y,z) outputs may have to be time multiplexed to the host microcomputer. Thus, fast time multiplexing is non-critical since most host computers can only load one point (8 bits) at a time anyway. The A/D time multiplexing may provide some slow down since at least two input values are required before processing can take place. Simple interrupts and acknowledgements can be established to

provide the required protocol.

The technology to be used is NT's 3 micron double metal CMOS. This should supply a relatively high packing density while providing sufficient speed. The silicon area to be employed is two of NT's A-size pad frames (A size = 4.5mm x 4.5mm). More area or ICs could have been used but this would result in an increase in size of the entire system, along with associating additional mask cost. Two A-size frames were decided upon, since one cannot satisfy the speed requirement and more than two will become too costly. Each A-size pad frame comes in a 40-pin package. The limit to 40 pins is a constraint which may affect the amount of busing between the two ICs.

Probably the most important factor is the required speed and accuracy. As was mentioned previously (Chapter 2), a 25 KHz TP throughput was desired. In order to perform 5 divisions, 4 multiplications, 6 add/subtracts and 1 square root operation in 40us, more than a single ALU is required. It was determined that separate multiplier, divider, and square root units were required, over and above simple add/subtract circuitry. The required accuracy was eight bits which is sufficient considering the uncertainty in the sensor (PSD). As discussed earlier, certain other techniques (bit-slice, large lookup table) could provide sufficient speed for eight bits, but given the rate of

advancement in technology, more than eight bits may soon be required for other sensors. The additional ICs using a bit-slice approach will make the system cumbersome. Therefore, the objective was to design an architecture which could be used later by simply increasing the silicon area to provide more bits, while still satisfying speed requirements.

3.2 Processor Architecture

In order to satisfy silicon area restrictions as well as speed requirements, a dedicated TP architecture was chosen. By using a dedicated architecture, instruction memory and decode circuitry could be avoided. The dedicated block architecture for the two ICs is shown in figure 3.1. There are separate divider, multiplier, and square root units, as well as 3 adders and 1 subtractor. These processing units provide sufficient performance to satisfy the speed and accuracy requirements.

3.2.1 System Pipeline

The overall vision system is pipelined with a depth of three. While one of the samples is being sensed in the PSD and converted to digital form, the previous sample is being loaded into the TP and the initial add/subtract operations

Figure 3.1 : Dedicated TP Architecture

are being performed. The second previous sample is being processed in the divider/multiplier stage on the TP. This pipeline provides greater data throughput since the total process time of the TP does not have to be added to the PSD rise time, and the A/D conversion period. In order to satisfy the required speed criterion of 25 KHz, the slowest section of the pipeline must complete its operation in 40us. It has already been shown that the PSD and A/D conversion can easily be performed within the 40 microseconds. The input stage and initial add/subtract within the TP will be quite fast since little processing needs to be performed. Therefore, the divider/multiplier section may be the section not satisfying the speed criterion since 5 divisions, 4 multiplications, 1 square root, and 2 additions must be performed within the 40us time frame.

3.2.2 Control States

The system clock for the TP is based upon the speed of the slowest unit. As will be shown later, the divider is the slowest device. Noting that 5 divisions are required, a minimum of 5 system states are needed. If more silicon area was available, several instances of the divider could have decreased the number of system states, but due to the silicon area constraint, only one instance of the divider,

multiplier, and square root unit was possible. Although 5 system states are the minimum required, 7 states were used. The two extra states can be explained. Due to the constraint of 40-pin packages, the two ICs forming the TP were joined by a bidirectional bus (divide bus). One of the extra cycles is due to this bidirectional bus transfer. If more pins were available, a separate bus could be used to regain this lost cycle. The other extra system state can be seen by examining the triangulation formulae. In order to calculate the x and y coordinates, two multiplications must be performed after the final division (final division calculates z) has been processed. The calculation of the y coordinates adds the extra state, while the x coordinate is calculated on the first system state of the next sample. This extra state is not recoverable and thus, only the bidirectional bus transfer state could be eliminated given different IC packaging. The unit responsible for providing the 7 system states is termed Shifter7 which is shown in figure 3.2. Shifter7 holds at state 7 until the host computer issues a start signal. Shifter7 then cycles through its seven states and halts at state 7. This will be discussed in more detail later in this chapter.

Note : All empty blocks are data flip-flops.

Figure 3.2.: Shifter7 Architecture

3.2.3. Input Stage

The input stage of the TP is contained entirely in IC1. This stage forms one stage of the pipeline and consists of an input control circuit (shifter5), four input registers (x_1 , x_2 , y_1 , y_2), and the initial add/subtract circuitry. Shifter5 receives as input, a STARTIN signal which takes place at one of the seven system states. The FCLK input to the shifter5 is 4 times faster than the system CLK signal. In this way, all four inputs are loaded into the TP during one system state even though the inputs are time multiplexed on the Input Bus. This input sample data then takes two cycles to be processed by the add/subtract circuitry. Therefore, the STARTIN signal can be issued on system state 5. The calculation for $(y_2 + y_1)$ and $(y_2 - y_1)$ can be performed in state 6 and $(x_2 + x_1)$ and $(x_2 - x_1)$ can be processed in state 7. Since only 3 of the 7 system states are required in this section of the pipeline, the system can be reconfigured to a pipeline of depth 2. In this way, the PSD, S/H units, A/D converters, and input stage of the TP would form one section of the pipeline. Of course this puts a more difficult time constraint upon the PSD, sample and hold units, and A/D converters without providing any more speed benefits. Therefore, the 3 section pipeline is more cost effective since slower devices can be used while not losing any

speed.

3.2.4 Divide/Multiply Stage

The majority of the silicon area on the two ICs which form the TP is used for the Divide/Multiplier section of the system pipeline. Within the seven system states, 5 divisions and 4 multiplications must be performed, and thus, these units must work in parallel. Following is a list of the exact operations which occur during the seven states.

State 1: Divider; $y'(t) = [y_2(t) - y_1(t)] / [y_2(t) + y_1(t)]$
Multiplier; $x(t-1) = z(t-1) * x'(t-1) / L$

State 2: Divider; $x'(t) = [x_2(t) - x_1(t)] / [x_2(t) + x_1(t)]$
Multiplier; $y'(t) * y'(t)$

State 3: Divider; calculates $y'(t) / L$
Multiplier; $x'(t) * x'(t)$

State 4: Divider; calculates $x'(t) / L$
Adder ; $y'(t) * y'(t) + x'(t) * x'(t)$
Square Root; $S(t) = \text{Sqrt}(y'(t) * y'(t) + x'(t) * x'(t))$
Adder ; $S(t) + L \tan \theta$

State 5: Data Transmission from IC2 to IC1
(ie. bidirectional bus transfer =
lost state)
 $S(t) + L \tan \theta$ transmitted

State 6: Divider; $z(t) = L(r - L \tan \theta) / (S(t) + L \tan \theta)$

State 7: Multiplier; $y(t) = z(t) * y'(t) / L$
Interrupt sent to host computer

During many of the states, at least two units are operating in parallel. In this fashion, only 7 states are required to process all the triangulation formulae. In

order to satisfy the 25 KHz requirement, the divider (system clock based upon divider) must process in less than $40/7$ us.

3.2.5 Output Stage

The final silicon area of the TP is utilized by the output stage which communicates with the host computer. At state 7, an interrupt is sent to the host computer, which notifies the computer that the processing will be complete after that state. Upon receiving an interrupt, the host computer can activate the SELZ and SELY signals to the TP in order to time multiplex the z and y coordinates from the output bus. The computer then enables the next laser diode and issues a start command. The TP, upon receiving the start signal, will cycle to state 1 where the x coordinate is then calculated. The host computer must wait until state 2 before activating the SELX signal to put the x coordinate data on the output bus. However, by noting that the multiply time is less than the system cycle time (based upon divider), the x coordinate data will be available before the start of state 2. Therefore, the x coordinate can be retrieved faster. By noting the cycle time (less than $40/7$ us), it is obvious that few microcomputers will have time to issue the SELX signal and retrieve the data off the bus in less than the cycle time. Therefore,

waiting for state 2 to complete poses little time loss to the host computer. State 7 is the logical state to issue the interrupt signal to the host computer. However, the option is there to issue the signal on any of the 7 states. This may be useful if the host computer responds to interrupts rather slowly. The interrupt could be generated earlier, so that the host computer can be ready when the processed data is available.

3.2.6 System Initialization

Upon system power up or reset [17], a number of initialization procedures must be performed. A power on reset (POR) signal forces both shifter5 and shifter7 to their highest states, 5 and 7 respectively. Thus, when shifter7 receives the start signal from the host computer, it will start at state 1 (similarly for shifter5). Three constants must also be loaded upon system power up. These are L , $L \tan \theta$ and $L(r - L \tan \theta)$. These constants are not hard coded into the TP in order to provide flexibility in the vision system's optical configuration. The radius of the light sources, along with their angle of projection can be changed while still employing the same TP. Thus, some degree of programmability is provided. The host computer utilizes the output bus, where it receives the (x, y, z) coordinates, to download the three constants. The host

computer first activates the CONSTLOAD signal to the TP. This opens a path from the output bus to the divide bus connecting IC1 to IC2. This CONSTLOAD signal also tristates the output of various devices (divider), along the constant's data path. The host computer then transmits one of the constants to the output bus. By activating one of SELL, SELL($r-L\tan\theta$), or SELL $\tan\theta$ signals, the correct constant can be clocked into its associated register. Following the same sequence, all three constants can be loaded. Once constant loading is completed, the CONSTLOAD signal must be deactivated.

3.3 Design and Simulation

Now that the architecture of the dedicated TP has been designed, the individual units must be further developed. This unit development should consist of architectural or circuit design, followed by some form of simulation. In this case, HILO was used for logical simulation, since it provided circuit extraction capabilities from a schematic editor. At this point, it should be noted that HILO simulation was based upon a 5 micron simulation library (3 micron was not yet available), and that wiring capacitances were not accounted for. A more detailed, layout simulation will be discussed later in this chapter. This logical

simulation, was more to determine the correct functionality of the various blocks, than to provide any critical timing data.

3.3.1 Divider

The choice of the divider is crucial since the system clock cycle is based upon the speed of the divider. Certainly, a serial or iterative divider will not provide sufficient speed. Therefore, a cellular array divider was chosen. The divider scheme should provide sufficient speed while minimizing silicon area. The silicon area can be reduced through fewer gates or a very regular structure. Table 3.1 summarizes a comparison of various array dividers [9][12].

By examining table 3.1, it can be noted that if speed was of greatest importance, a nonrestoring array divider with carry look ahead (CLA) would be chosen. Due to its irregular structure (3 different functional blocks used many times), the silicon area would greatly increase despite its gate count being comparable to other slower methods. In fact, due to the area constraint, the nonrestoring divider with CLA had to be eliminated due to its excessive silicon area. The gate counts must be readjusted to coincide with the CMOS technology. In table 3.1 the gate count is often based on some AND and OR gates.

<u>Type of Array Divider</u>	<u>Gate Count</u>	<u>Divide Time</u>
Restoring (Dean)	$14n^{**2}$	$(3n^{**2}-1)D$
Nonrestoring (Guild)	$17(n+1)^{**2}$	$(3(n+1)^{**2})D$
Nonrestoring with 2-level CLA and carry- save (Cappa-Hamacher)	$1+(n+1)^{*}$ $(18n+Sqroot(n)+1)$	$(11n+12)D$
Nonrestoring with 1-level CLA and carry- save (Cappa-Hamacher)	$18n^{**2}+28n+11$	$(9n+10)D$

where n is the number of bits and D is a unit gate delay.

Table 3.1 : Divider Comparison

Of course in CMOS, NAND and NOR are used and this will increase the gate count of each. Therefore, for Restoring (Dean), the gate count formulae for CMOS becomes $24n*n$ (for $n=8$, gate count = 1536), and for Non-restoring (Guild), the new formulae becomes $11(n+1)*(n+1)$ (gate count for $n=8$ is 891). Considering the difference in gate count and the stringent silicon area restrictions, Non-restoring (Guild) cellular array division was chosen. Its schematic diagram is shown in figure 3.3.

It must now be shown that the chosen divider provides sufficient throughput to satisfy the speed criterion. Using the estimate of $243D$ where D is a unit gate delay, and adding a wiring capacitance of $1pF$ on each gate, $243D = 243 * 4ns = 972ns$ for the total divide time. This is not fully accurate since the extra gates due to the CMOS technology were not accounted for. Noting that within each cell of the array the longest path is the carry generation, the total divide time is 64 times the carry generation speed. The carry generation speed is the propagation of 1 XOR, 3 inverters, 2 NOR, and 1 NAND gates. Utilizing a $1pF$ load per gate, this results in $6ns + 3(4ns) + 2(8ns) + 6ns = 40ns$. Thus, the total divide time is $64 \times 40ns = 2.56$ microseconds. The required divider speed is $40/7$ us which is almost double the 2.56 microseconds. Also noting that a $1pF$ load on each gate in an array type

Note : All cell blocks are CAS cells.

Figure 343 : Divider Architecture

structure is a pessimistic estimate, the divider will have little-trouble in satisfying the speed requirements. Due to the divider's high gate count, the simulator HILO was not capable of simulating the entire divider. Therefore, a functionally equivalent 2-bit version was simulated. The results are shown in figure 3.4. The simulated division was $A7A6/B7B6 = Q7Q6$. Note that for decimal results or dividing by zero, $Q7Q6 = 11$ in binary. This can be detected as an error by taking the exclusive-or (XOR) of $Q6$ with the carry in to the $A6/B6$ cell.

3.3.2 Multiplier.

A suitable multiplier architecture must be chosen. The multiplier must have a throughput equal to, or greater than the divider while minimizing silicon area. Serial multiplication cannot satisfy the throughput requirement, and thus, a cellular array multiplier was used. Table 3.2 summarizes a comparison of 4 common array multipliers [12].

From the table, it can be noted that each multiplier performs relatively evenly in terms of throughput. Due to the regular structure of the Baugh-Wooley method (all Type 0 full adders), this method reduces silicon area. Therefore, the Baugh-Wooley method of multiplication was chosen. Its associated architecture is shown in figure 3.5.

Figure 3.4 : 2-Bit Divider

m by n Two's Complement Array Multipliers

<u>Full Adder</u>	<u>Pezaris</u>	<u>Tri-Section</u>	<u>Bi-Section</u>	<u>Baugh-Wooley</u>
Type 0	$(m-2)(n-2)$	$(m-1)(n-2)/2$	$(m-2)(n-1)$	$m(n-1)+3$
Type 1	$m-2$	$(m-1)(n-2)/2$	0	0
Type 2	$m+n-3$	$2(m-1)$	$2(m-1)$	0
Type 3	1	0	0	0
Delay	$2(m+n)D-2D$	$2(m+n)D-2D$	$2(m+n)D$	$2(m+n)D$

where D is a unit gate delay,

type 0 adders perform $x+y+z=+C,+S$

type 1 adders perform $x+y-z=+C,-S$

type 2 adders perform $-x-y+z=-C,+S$

type 3 adders perform $-x-y-z=-C,-S$

and C represents the carry and S represents the sum.

Table 3.2: Multiplier Comparison

Type 0 Full Adder

Note : All cells are Type 0 Full Adders.

Figure 3.5 : Multiplier Architecture

Simulation using HILO was performed on a 3-bit model of the multiplier in order to test its functionality. Simulation results are shown in figure 3.6. Table 3.3 summarizes the test vectors and results. Thus, the functionality of the Baugh-Wooley multiplier is confirmed.

By examining the multiplier architecture in more detail, it can be shown to have a delay of $15 * (\text{carry generation delay}) + 7 * (\text{sum generation delay})$. By noting that the carry generation using a 1pF load on each gate is 40ns and the sum generation is 12ns, the total multiplier delay is 0.684us. This throughput rate more than satisfies the requirement of being faster than the divider.

3.3.3 Adder Architecture

An adder architecture must be chosen which minimizes silicon area while completing its processing within one system cycle. Considering the length of a system cycle (based upon the divider speed), a simple ripple carry adder should provide sufficient processing power while minimizing area. Figure 3.7 shows the ripple carry architecture. Simulation using HILO was performed upon the adder to test its functionality. The simulation results are shown in figure 3.8 and 3.9, and summarized in Table 3.4 confirming the adders functionality.

The speed of the adder is the number of bits minus one

Figure 3.6 : 3-Bit Multiplier

Multiplier Test Vectors and Results

<u>M * Q (3-bit binary)</u>	<u>Results (6-bit binary)</u>
000 * 000	000000
111 * 000	000000
111 * 111	110001
110 * 110	100100
010 * 101	001010

Table 3.3 : Multiplier Simulation Results

Note : All cells are Full Adders.

Figure 3.7 : Adder Architecture

Figure 3.8 : 8-Bit Adder

Figure 3.9 16-Bit Address

8-Bit Adder Test Vectors and Results

<u>A + B (8-bit binary)</u>	<u>Results (9-bit binary)</u>
00000000 + 00000000	000000000
01010101 + 00000000	001010101
01010101 + 01010101	010101010
11111111 + 11111111	111111110

Table 3.4 : Adder Simulation Results

(n-1) times the carry generation (CG) speed plus a single sum generation (SG).

$$\text{Adder Result} = (n-1)\text{CG} + \text{SG}$$

For a 1pF gate load, CG was found to be 34ns (similarly with divider and multiplier), and SG was 10ns. Therefore, the speed of the 8 and 16 bit adders is 248ns and 520ns respectively. Both of these throughput rates are well within the system cycle time.

3.3.4 Subtractor Architecture

The subtractor must satisfy the same constraints as the previous units. In order to choose an appropriate subtractor and its surrounding circuitry, the other units must be fully understood, particularly with their treatment of signed numbers. All of the previously discussed units (divider, multiplier, adder) process unsigned numbers, and thus, if for this particular application negative results are possible from the subtractor, this data cannot be sent to any of these unsigned processing units. Noting that the subtractor performs the calculations $(x_2 - x_1)$ and $(y_2 - y_1)$, and that the sensor may output a larger x_1 or y_1 than x_2 or y_2 respectively, the subtractor may have a negative result. Therefore, this negative result must be further processed in order to be compatible with the unsigned units. One

possible solution, and the one which was implemented, is to perform two's complementation when a negative result occurs. The new positive number can then be processed and restored to its negative form by another two's complementation. The circuits for the complementation are shown in figures 3.10 and 3.11. SAS (sign adjust circuit) performs the initial two's complement, and TSAS (terminating SAS) processes the final complementation just before the data is available to the host computer. The associated functional HILO simulations for SAS and TSAS are shown in figures 3.12 and 3.13.

In choosing a suitable subtractor, the subtract time plus the two's complement time must occur within one system cycle. Using a ripple carry subtractor as shown in figure 3.14, the required speed can be achieved while minimizing silicon area. The functional HILO simulation is shown in figure 3.15. Table 3.5 summarizes the input test vectors and results.

The actual propagation delay for the total subtract time plus the sign adjust time is relatively equivalent to an 8-bit addition time plus the SAS time of $6 \times 16\text{ns}$. Therefore, the total subtract time including sign adjustment is $248\text{ns} + 96\text{ns} = 344\text{ns}$, which is still far less than the system clock cycle.

Note : All cells are Two's Complement Cells

Figure 3.10 : SAS Architecture

Note : All cells are Two's Complement Cells

Figure 3.11 : TSAS Architecture

Figure 3.12 : SAS

Figure 3.13 : TSAS

Note : All cells are Full Adders with one input inverted.

Figure 3.14 : Subtractor Architecture

Figure 3.15 : Subtractor

Subtractor Test Vectors and Results

<u>A - B (7-bit binary)</u>	<u>Results (6-bit binary)</u>
0000000 - 0000000	000000
1111111 - 1111111	000000
0111111 - 0000001	011110
0111111 - 1000000	111111

Table 3.5 : Subtractor Simulation Results

3.3.5 Square Root Architecture

Typically, square root operations are performed iteratively using an ALU and perhaps some other processing units. Using an iterative approach, a single 16-bit square root cannot be performed within the required system cycle given the silicon area available. Therefore, an array square root circuit was chosen. The circuit architecture is shown in figure 3.16. Note that the functional building cell is identical to the cells used to form the divider.

Functional simulation using HILO on a 4-bit square root circuit is shown in figure 3.17. Table 3.6 summarizes the test vectors and the results confirming the circuit's functionality. From the architecture, the circuit's speed can easily be seen to be $(9*8*7*6*5*4*3*2)$ (speed of one cell). Previously (divider section), the speed of each cell was shown to be 40ns with 1pF gate loading. Therefore, the total square root time is 1.76us. Note that in the system architecture, a 16-bit addition, a square root, and an 8-bit addition must be performed within state 4. Of course, the bidirectional bus transfer in state 5 will not occupy the whole cycle and thus, the processing of state 4 could, if required, overlap at least halfway into state 5. The add-sqroot-add delay time is $520\text{ns} + 1.76\text{us} + 248\text{ns} = 2.528\text{us}$. The divide time, and thus, the system state time is 2.56us. Therefore, there will be no

Note : All cells are CAS cells.

Figure 3.16 : Square Root Architecture

Figure 3.17 : 4-Bit Square Root

Square Root Test Vectors and Results

<u>Sqroot(A) (4-bit binary)</u>	<u>Results (2-bit binary)</u>
0000	00
1001	11
0001	01
0101	10
0011	10

Table 3.6 : Square Root Simulation Results

problems completing the required processing in the available time.

3.3.6 Control Circuits

The function of the control circuits is to supply the 7 system cycles as well as the 4 input cycles for the loading of input data. In both cases a shift register is used to provide the different states. Shifter5 provides the 4 input states. There are three inputs to this circuit (POR, FCLK, and STRT). The power on reset (POR) sets the control shifter to its highest state where it holds until a STRT signal is received. The control shifter waits until the second clock edge following the STRT signal to ensure clock synchronization. The shifter then cycles through its 4 states and holds again at its highest state. Shifter7 which supplies the 7 system cycles, acts identically except that it provides 7 states instead of four. The functional HIL0 simulation for both shifter5 and shifter7 are shown in figures 3.18 and 3.19 respectively.

3.3.7 Registers

All standard data registers are based upon simple data flip-flops with no set or reset. Set and reset are not required since the first valid data clocked in is the first data used. Figure 3.20 shows the HIL0 simulation for an

Figure 3.18 : Shifter5

Figure 3.19 : Shifter7

Figure 3 20 REGD driven by TESTREGD (file regd)

Figure 3.20 : 8-Bit Register

8-bit register where each data bit receives the IO signal.

3.3.8 Bus Arbitration

In determining which piece of data is accessing a particular bus, a circuit for enabling and disabling data transfer to and from a bus must be designed. A simple transmission gate turned on and off at the suitable time will provide this ability while occupying little silicon area. The functional HILO simulation for an 8-bit transmission gate (Pass8) is shown in figure 3.21. Bit P0 receives input PIO while all other bits receive input PI1.

3.3.9 Summary of Delays

Table 3.7 shows a summary of individual part delays.

3.4 Architecture Simulation Using SPLICE

Once each individual part had been functionally simulated, an overall block architecture simulation with approximately correct timing values could take place. SPLICE was used as the block architecture simulator. Due to the limitations of SPLICE, the architecture was divided into 5 main component blocks; as shown in figure 3.22. The control block encompasses both control shift registers and receives external signals POR and STCON. The clock speed

Figure 3.21 : Bus Arbitrator

Summary of Device Delays

<u>Part</u>	<u>Delay (ns)</u>
Divider	2560
Multiplier	684
8-bit Adder	248
16-bit Adder	520
Subtract with SAS	344
Square Root	1760
Add-Sqroot-Add	2528

Table 3.7 : Part Delays

Figure 3.22 : SPLICE Architecture

is built into the simulation. The input-divide block receives the four inputs (IN1, IN2, IN3, IN4) along with encompassing the initial Add/Subtract circuitry and the divider and its associated transmission gates and registers. The Multiply block includes the multiplier and its surrounding circuitry. The Add-Sqroot-Add block performs the 16-bit addition, followed by the 16-bit square root, and finally the 8-bit addition. This data is then fed back to the divider. The Output stage contains the output circuitry which communicates with the host processor. The signals to be noted in this simulation are: the four input signals, IN1...IN4; the external signals to the control block, POR and STCON; the three host processor signals to the output stage, SELZ, SELY, and SELX; the system cycle control bus, CONBUS; the output of the divider, DIVBUS; the output of the multiplier, MULTBUS; the output of the Add-Sqroot-Add block, S; and the output data sent to the host processor, OUTBUS. It is assumed that the three required constants are already downloaded and available in the system.

SPLICE had the limitation of a maximum of nine models. The dedicated TP used five models since the architecture could be logically separated in this fashion. Most models actually encapsulate several units. The actual SPLICE models can be found in appendix A. Buses were modelled as

analog signals and thus, the analog voltage level represents the decimal equivalent bus value. In this way, a single line or wire was used to represent a bus.

Figure 3.23 shows the simulation results for the control block. When the POR signal is received, the output of the state shifter (CONBUS) is set to state 7. State 7 is held until the STCON signal is received, at which time the shifter cycles through its 7 states and returns to hold at state 7. Inputting the four digital signals from the A/D converters in the vision system is shown in figure 3.24. The decimal values of the 8-bit numbers is displayed on the y-axis. The data was loaded during state 5. The first input occurred at 33us. The 7 state shifter output, CONBUS, can also be seen cycling through its states.

Figure 3.25 shows the results of the internal units and buses simulation. The 7 state controller can be observed, as well as the divider, multiplier, and Add-Sqroot-Add circuit outputs. The particular state at which an output of a unit changes shows that the unit was active. Note that the signal S, is actually $S + L \tan \theta$ from the triangulation formulae. The data from all the units change at the end of a particular state so that it is available for the next state. In the case of the multiplier, the output actually changes approximately halfway into the state, but since the output is sampled at the beginning of

Figure 3.23 : Control Block

Figure 3.24 : Voltage Inputs

Figure 3.25 : Internal Units and Buses

the next state, this is immaterial. The final block to be analyzed is the Output block. Its associated simulation plot is shown in figure 3.26. When state 7 is reached, an interrupt is sent to the host processor. The host then issues a SELZ signal followed by a SELY signal. The appropriate data is sent on the OUTBUS. The host processor then issues a STCON signal in order to restart the control shifter. Once state 1 is completed, the SELX signal can be activated and the x coordinate is placed on the OUTBUS.

The actual output values and data flow can be accurately analyzed from the output data files. The results were correct in all instances. The TP architecture performs all the necessary triangulation formulae on the associated correct state. Therefore, since the TP architecture has been simulated and the individual parts have been designed and simulated, the layout of the TP on silicon is the final step.

3.5 Layout of the Dedicated TP

3.5.1 CAD Tools

In order to design a functionally correct layout for fabrication, several design tools are essential. A layout or mask editor, or a silicon compiler is required to perform the layout artwork, including both interconnections

2 of/de 2

Micro-D

Figure 3.26 : Output Block

and devices. A means of checking the layout in order to satisfy process limitations or rules is required.

Extraction and simulation of the layout should be performed to establish functional as well as timing correctness. For a large layout, some parts of the simulation can be avoided if the layout netlist can be shown to be equivalent to a previously logically simulated schematic netlist. All of these tools for layout artwork, design rule checking, extraction simulation, and netlist comparison form a core of essential tools resulting in a complete design environment. An environment lacking any of these CAD tools will be hard pressed to guarantee the layout correctness. Unfortunately, due to the economics of such tools, a complete design environment is not always available. Some of the core tools were not available for the layout of the TP.

The artwork for the TP was performed utilizing a symbolic layout editor, VALE, on a Metheus workstation, and a silicon compiler, CHIPSMITH, on a SUN workstation[6]. CHIPSMITH provides a fast layout which automatically corresponds to the schematic (netlist comparison not required). Unfortunately, the layout from CHIPSMITH consumes more area than one completed through VALE. Therefore, a mixture of the two tools was required to fit necessary components onto the fixed silicon area. Design

rule checking (DRC) was accomplished either through the work stations or the Canadian Microelectronics Corp. (CMC) itself. Thus, the DRC tool was available. The main lack of tools occurred in the simulation and netlist comparison area. CHIPSMITH automatically provided netlist comparison while VALE did not. No other netlist comparison tool was available. Therefore, all layout functional and timing verification was forced upon the simulators. Two types of simulators were available after layout extraction: a transistor level simulator, SPICE; and a switch level simulator, Hg. Hg provided accurate simulation for small blocks such as the control shifters, the registers, and the bus arbitrators (transmission gates), but it failed on other circuits. The reason for this failure was due to the size of the other blocks as well as the extensive use of transmission gates within these blocks. Switch level simulators commonly fail on transmission gates, and since the XOR gate used was based on transmission gates, the divider, adder, subtractor, sign adjustor, multiplier, and square root circuits could not be simulated with Hg. A version of SPICE called HSPICE was used so that larger circuits could be accommodated. Unfortunately, due to the large transistor count, an 8-bit adder required two days to simulate and even then only one bit of the result could be recovered. This vast computing time forced an immediate

abandonment of HSPICE. Therefore, with no real means of simulating the layout or comparing it with the previously functionally simulated schematic, the fabrication results could not be guaranteed. Since manual (by eye) checking was used for interconnectivity, it is likely that several iterations will be required for a completely functional TP.

3.5.2 Standard Cells

The layout itself is based upon standard cell design. A previously designed standard cell library of common gates and flip-flops was available for use with the layout editor VALE. The delays of these gates were built into the functional simulation using HILO in some instances. A new smaller set of standard cells was designed to be compatible with CHIPSMITH. These were used extensively in the second IC, but were only available towards the end of the design cycle for the first IC.

3.5.3 Simulation

Four of the circuits were simulated successfully after layout extraction using Hg. (Fortunately, two of the circuits are the control shifters, and thus, the integrity of the control circuits was verified.

3.5.3.1 Control Shifter Simulation

Figures 3.27 and 3.28 show the Hg simulation for shifter5 and shifter7 respectively. Both shifters act similarly and thus, only shifter5 will be discussed. When the POR signal goes low upon reset, the shifter is set to its highest state (e). A start signal (STRT) notifies the shifter to begin cycling through its states. The shifter begins its cyclic pattern only after two rising clock edges (FCLK) to ensure clock synchronization. Once the shifter cycles through its states, it holds again at its highest state until a start signal is received again.

3.5.3.2 Register Simulation

The Hg simulation for an 8-bit register is shown in figure 3.29. Only one bit of the input and output is shown although all bits acted similarly. Upon the rising edge of the clock signal (ck), and after a suitable delay, the data (d0) was clocked into the register (q0).

3.5.3.3 Bus Arbitrator Simulation

Figure 3.30 shows the Hg simulation of the 8-bit transmission gate (Pass8) which is used for bus arbitration (only one bit shown). It is evident that when the clock (ck) is high, the output (y0) follows the input (d0), and when the clock is low, the output holds its value.

Figure 3.27 : Hg Shifter5

Figure 3.28 : Hy Shifter7

Figure 3.29 : Hg Register

Figure 3.30 : Hg Bus Arbitrator

3.5.4 System Layout

Figures 3.31 and 3.32 show the layout of the two ICs which form the TP. Figure 3.33 shows the basic interconnectivity of the two ICs. Both ICs were manufactured by Northern Telecom (NT) and the Canadian Microelectronics Corporation (CMC).

3.6 Test Results

Testing was performed on IC1 of the dedicated TP. This IC lacked a complete design environment since CHIPSMITH was not yet available. Two errors on the chip resulted in the impossibility of further testing. The input/output (IO) pads used were taken from the standard cell library. Unfortunately, these cells had not been previously tested, and thus, a design error existed. This error resulted in the impossibility of loading data through these pads. This could be alleviated by replacing these pads with newer pads which have been designed. The second error occurred with the power on reset (POR) circuitry. Upon power up, the control shifters were initialized to their correct states. Unfortunately, after the POR signal was removed, a floating node did not disable the set and reset signals to the control shifters. Thus, the shifters were stuck at their initial state. This problem would be eliminated by the removal of the transmission gate which

Figure 3.31 : IC1 Layout using VALE and CHIPSMITH

Figure 3.32 : IC2 Layout using CHIPSMITH

the POR signal controls. This form of error is difficult to detect from simulation. The inability of the control shifter to get out of its initial state resulted in the impossibility of loading or processing data further.

By resolving these two problems, IC1 can be resubmitted for fabrication. IC2 is currently being manufactured with the new IO pads. IC2 was also produced using CHIPSMITH, which should guarantee the integrity of the interconnectivity.

3.7 Improvements

There are four main areas where the dedicated TP could be improved. Simply by increasing the PIN count, the bidirectional bus could be replaced by two separate buses. This would remove one of the system states and thus, increase the throughput of the TP. With more silicon area available, several instances of the divider and multiplier could be used. This would allow for a totally pipelined system, where the speed of the pipeline would be governed by the rise time of the PSD (8us). More accuracy in terms of increased bits could be provided given more silicon area. This would only be necessary if the accuracy or size of the PSD were improved. The final area for improvement is functionality. Currently, the TP is dedicated for use

with a particular PSD. Even though the constants are programmable, the triangulation formulae could also be made programmable. This would allow for different types of PSDs or other sensors in general. Such a general purpose solution will be proposed and discussed in chapter 4.

3.8 Applications

Since the developed triangulation processor is a dedicated architecture, its only application is its use with the previously discussed vision system or other systems requiring the exact same formulae processing (unlikely to find any).

3.9 Summary

The TP needed in the implementation of the previously proposed vision system has been developed. A stringent set of design objectives and constraints for the TP has been satisfied by a dedicated architecture. The TP has then been taken through the full developmental process of logical or functional design and simulation, layout design and simulation, fabrication, and testing. Four areas of improvement have been proposed, one of which will be further developed in chapter 4.

CHAPTER 4: GENERAL PURPOSE PROCESSOR

4.0 Introduction

In chapter 3, a dedicated TP was presented. The main limitation of such a processor is its lack of programmability. Thus, the dedicated processor cannot be used for other applications, be they vision systems or other sensors. Recently, new faster and cheaper PSDs have been developed but a different set of formulae are required. The dedicated TP will not work but the general purpose TP could be reprogrammed. Of course, the main reason for developing a general purpose TP would be to increase the number of applications which could be satisfied. Due to these increased users, many more ICs would be produced. Economics of scale make this a more attractive product. Therefore, this chapter will propose and develop a general purpose TP which can be used by most vision systems requiring triangulation calculations. Also, this general purpose processor can be used for any application requiring 8 to 12 bits of accuracy. This accuracy satisfies the majority of different sensor applications.

4.1 Design Objectives and Constraints

The general purpose triangulation processor (GPTP) must be able to satisfy the requirements of the dedicated TP as well as any other foreseeable criteria for a wider range of applications. The first requirement is that the processor's speed have sufficient capabilities to satisfy most targetted applications. Given that there are a number of relatively fast general purpose DSPs available on the market today, this TP should probably be faster if it plans to cut a niche in the market. The processor is aimed at low accuracy (8 to 12 bits) applications. In order to satisfy higher accuracy requirements, the architecture of the GPTP would have to be slightly changed. This will be examined in detail in the Improvement section (4.5). The GPTP is aimed at a single chip solution in order to reduce manufacturing costs as well as increasing the speed of the processor. Obviously, Northern Telecom's 3 micron technology is not capable of providing the necessary silicon area, speed, or feature size needed to satisfy the above requirements. In fact, a 1.25 or 1.5 μm technology with a 1cm by 1cm die is probably most suitable. The smaller technology coupled with the increased silicon area will be more costly than that of the dedicated TP, but only one set of masks rather than two will be required. Due to the technology difference, the GPTP will be much faster

than the dedicated TP. The exact speed difference will be determined by the GPTP's architecture.

4.2 Processor Architecture

In order to manufacture a competitive product (GPTP), it was deemed necessary that the GPTP should provide comparable or better data computation throughput when compared with other available general purpose single chip DSPs. This speed comparison should take place at the architecture level so that technology differences are not accounted for. The main reason for this is that the GPTP could be produced in the same technology as the fastest DSP if it were manufactured by the same organization. The GPTP architecture is shown in figure 4.1. The architecture will now be further broken down and analyzed.

4.2.1 Processing Units

In order to satisfy the typical mathematical operations required for triangulation formulae, multiplication, division, subtraction, addition, and square root processing is necessary. Trigonometric lookup tables are not needed and in fact, are not very feasible. The trigonometric formulae can be processed by the previously mentioned mathematical operations. The lack of feasibility of such tables is evident since for a general purpose

Figure 4.1 : GTPP Architecture

solution, all the different trigonometric tables with inverses would be required. From figure 4.1, the mathematical processing units can be observed. Separate division, multiplication, square/square root, and ALU units are provided. This can be accommodated on a single chip due to the low bit accuracy of these units. It is evident from the number of processing units, that if each unit can compute in parallel with no busing bottlenecks, then the GPTP will compute faster than most available single chip DSPs. Assuming an 8-bit system accuracy, the divider, multiplier, and squarer circuits require 8-bit operands, while the square root and ALU circuits require 16-bit operands. The reason for the 16-bits is to accommodate the output of the multiplier and squarer (16-bit output) which often needs further ALU and square root processing.

The divider is the slowest of the four processing units. It takes two system cycles to complete its processing. Most likely a non-restoring array divider with one level CLA and carry-save would be used to increase the computational speed. The estimated speed is $(9n+10)D$ where D is a unit gate delay [12], and n is the number of bits. The multiplier requires one system cycle, and thus is twice as fast as the divider. Most array multiplier configurations will provide this speed. For example, the Baugh-Wooley 8-bit multiplier of the dedicated architecture

has an approximate delay of $2(m+n)D$ where D is a unit gate delay and m & n are the number of bits of the two operands [12]. Therefore, for an 8-bit divider and an 8-bit by 8-bit multiplier, the respective delays are $82D$ and $32D$.

Most sets of triangulation formulae as well as any trigonometric calculations require the square root operation. This operation could be performed by the other units but this would occupy several system cycles as well as utilizing these other processing unit's resources.

Therefore, as with the dedicated TP, a square root unit is provided. With little additional circuitry, the squaring operation can also be included (useful when processing sum of squares, etc.) [12]. The schematic of such a unit is shown in figure 4.2. Typically, such a unit is faster than a divider but not twice as fast. Thus, for a safe design and utilizing the fact that not many square/square root operations will be required, two clock cycles are reserved for this unit. The unit will accept either an 8-bit operand for squaring or a 16-bit square root operand. The outputs will be 16 and 8 bits respectively.

The final element of the four processing units is the ALU. This is a 16-bit unit so that outputs from the multiplier and squarer can be accommodated. The operations which can be processed by the ALU are as follows: addition, subtraction, NOP, compare, pass operand A, shift

Note : All cells are CASF cells unless otherwise noted.

Figure 4.2 : Square/Square Root Architecture

1-bit left, shift 1-bit right, negate, increment, and decrement. Using a carry look ahead (CLA) configuration, the ALU will actually be the fastest of the four processing units, and thus will complete its processing in a single system cycle (16-bit add or subtract time with carry or borrow flags set is 14D [12]). Therefore, if these four processing units can truly compute and access data in parallel, two system cycles, which is equivalent to two user instructions, will result in 1 division, 2 multiplications, 1 square or square root operation, and 2 ALU functions. Given today's fast technology, an 8-bit division (two system cycles) should easily be obtainable in 30 to 100 ns and perhaps faster. The GTP architecture must now be further examined to ensure the parallel operation of the four processing units.

4.2.2 Instruction Cache

From figure 4.1, the instruction cache, instruction register (IR), program counter (PC), and jump circuitry is evident. The instruction cache accommodates up to 32 instructions. In order to control all 4 processing units with a single instruction, the instruction word length is 64-bits. The instruction cache can be preloaded (8 bits at a time) through the external busing. Typical sensor applications will load the cache initially and then leave

it, since 32 instructions can perform a large amount of processing and most sensor tasks are repetitive. For the previously discussed triangulation formulae, only 11 instructions were required. Thus, these eleven 64-bit instructions would be preloaded and then continuously looped through. The instruction cache is actually divided into a 32-bit by 64 word configuration. Thus, each 64-bit instruction is divided into two equal parts. This is done to reduce busing and other circuitry. A 32-bit IR is all that is needed. The instruction cache outputs one 32-bit segment of an instruction at a time. This is done by dividing the system clock in two. Upon the start of a system cycle, the first instruction segment is made available to the various processing units. This first 32-bit segment selects the input operands and functions of the four processing units. During the first half of the system cycle, when this first instruction segment is being utilized, the second segment is being accessed in the instruction cache. Thus, halfway into the system cycle, the second instruction segment is made available through the IR. The second 32-bit segment controls output enable and output destinations of the four processing units as well as any jump commands. During the second segment (second half of the system cycle), the next instruction is being decoded and accessed in the cache. Thus, there

exists a pipeline with a depth of two in the instruction cache. The use of two 32-bit instruction segments greatly reduces the busing since both segments can utilize the same bus except for some minor local routing surrounding the processing units.

The PC is a 6-bit counter which gets incremented twice every system cycle. The second segment of the instruction may request a jump operation. The PC is thus reloaded halfway through the instruction cycle. This allows the new jump address to be decoded in time for the next system cycle. Therefore, jump instructions constitute no additional time overhead. This is also somewhat due to the fact that all error flags are set well before the end of a system cycle.

4.2.2.1 Instruction Set

The control of this processor (GPTP) is performed mainly through the instruction set, the input circuitry, and the output circuitry. The 64-bit instruction word can be reduced to its appropriate fields. The first 32-bit segment consists of: (0..7) select divider operands; (8..15) select multiplier operands; (16) select square or square root function; (17..21) select square/square root operand; (22..27) select ALU operands; (28..30) select ALU function; and (31) external interrupt bit. The 32-bit

output (second) segment consists of: (0..5) divider output enable and destination; (6..10) multiplier output enable and destination; (11..16) square/square root output enable and destination; (17..21) ALU output enable and destination; (22) unconditional jump; (23) jump on error from the divider; (24) jump on carry set from the ALU; (25) jump on borrow set from the ALU; (26..30) new jump address; and (31) external interrupt bit. One can see that the majority of the instruction word is involved in selecting the input and output addresses for the four computational units. The supplied ALU operations are addition, subtraction, NOP, compare, pass operand A, shift 1 bit left, shift 1 bit right, negate, increment, and decrement. Note that the three ALU function select bits do not include the NOP and compare operations. NOP is performed by disabling the ALU's output, and the compare is performed by checking the borrow flag after a subtract and disabling the ALU's output. The jumps occur in the second 32-bit instruction segment, and thus occur in the latter half of the system cycle. At this time, the program counter holds the address of the next instruction. This address is starting to be decoded in the cache in order that it is available on the next system cycle. As soon as the jump is detected, the PC is reloaded with the new jump address. This address is then decoded during the latter half of the

system cycle. Therefore, jumps cause no additional delays. Typical applications will loop through the same set of instructions for each new data set from the sensor or input device. Therefore, once a set of data has been processed, a jump to the start of the program can occur.

4.2.3 Data Memory, Busing, and Address Decoding

With four separate on chip processing units, the GPTP is basically a single chip multiprocessor. Typical multiprocessor based systems have a communications overhead associated with data transfer or memory access. This in effect, reduces some of the parallel computational power of the individual processors. Since relatively small word lengths (8 to 12 bits) are being utilized, a special memory and busing technique could be employed to eliminate memory or communication bottlenecks.

From figure 4.1, the data registers, buses, and decoders can be observed. The data memory is made up of four register banks. Each bank contains eight 8-bit registers and has an associated decoder. In order to minimize circuitry and the instruction word length, all the data registers are not available as inputs to all four processing units. Bank 1 is available as input for the divider and multiplier units. Bank 2 is available to all four units. Bank 3 is available for the square/square root

unit, and Bank 4 is used by the ALU. This effectively gives 16 input registers to each unit (4-bits decoded). All the units can write to any of the 32 registers (5 bits). For some large data abundant applications, this memory may not be sufficient, but for most sensor applications such as triangulation formulae processing, 32 registers provides ample data space.

What makes this data memory special is the busing and decoding technique. When the first segment of an instruction is received, the input operands' addresses for the four units are decoded by the units themselves (segmented microprogramming [24]). Thus, each processing unit has a 4-bit decoder associated with each of its input operands. Through a network of switching circuits (tristate devices enabling and disabling which register has access to an operand bus), the appropriate data operands can be accessed (selected) by all four unit simultaneously. The 5-bit address decoder associated with each of the register banks is used to determine which register will be written by the processing units. These addresses are supplied during the second half of the system cycle by the second 32-bit instruction segment. Through a similar network of switching devices, a path is opened for the designated output register. At the start of the next system cycle, these registers will be latched with the

output data from the four processing units. To send the data from the units to the data registers, a separate output bus for each unit is required. Thus, the 56-bit Output Data Bus contains the results from all four computational units. The decoders also detect a write contention error. This is useful when debugging program code since for typical applications, the user should know where all data is being sent.

Now it is probably more evident why the bit accuracy (8 to 12 bits), and data memory size must be limited. The busing and switching circuits would encompass far too much silicon area if either the accuracy or data memory size were increased. By utilizing the lower accuracy and the specialized busing and decoding, it is evident that each of the four units can read and write to the data memory at the same time. Thus, truly parallel multiprocessor operation is achieved with no communication or memory time overhead. This parallel read and write is somewhat dependent upon the application. The GPTP is aimed at small (32 instructions) sensor applications where the user can easily avoid write collisions. For larger applications where write collision avoidance cannot be guaranteed, further precautions will be required.

4.2.4 Input and Output Stage

Now that the GTP has been shown to process data efficiently, the IC must be capable of effective communications with the external environment. This communication occurs in either the input stage or output stage of the processor. Typically, for inputting data to the data memory, the input address bus and input data bus can be utilized. Upon the next system clock (next instruction), the data will be latched into the associated register. The decoders associated with the register banks select the appropriate location. If a write contention error is flagged, then the user will have to wait for the next system cycle since one of the four processing units was accessing (writing) the same memory location. Thus, input communications is performed synchronously with the system clock.

The output stage acts asynchronously. By selecting the appropriate register through the output address bus, a switching network places the data on the output data bus. The user may want to ensure that this occurs shortly after a new system cycle to affirm data integrity. By allowing input and output communication via separate buses while the GTP is still processing other data, the throughput of the TP may not be effectively reduced. Of course, if large amounts of data loading is required, or all the previous

data is needed right to the last instruction of a program loop, then the GTP must have the ability to halt its operation. This is probably the typical method of communication since it ensures no overwriting of data. More specifically, an external interrupt bit can be sent by an instruction. This bit can be connected to the STOP pin. Thus, after completion of the current instruction, the processor will halt. This will allow time for outputting the processed data and inputting the new data. A host processor or simple external circuitry can be used to issue a RESET signal which allows the GTP to continue its operation. Alternatively, the external interrupt bit could be sent directly to a host processor which could then control the STOP, START, and RESET inputs. This method of halting the processor ensures data integrity but results in an overhead associated with the communication of the input and output data.

4.3 Triangulation Formulae Performance

Eleven system cycles were required for the GTP to process the triangulation formulae. Following is a list of exact operations which occur during the 11 cycles. Note that the division and square/square root time is two cycles.

State 1: Divider; $z(t-1) = L(r-L\tan\theta)/(S(t-1)+L\tan\theta)$
 ALU; $y_2(t) = y_1(t)$

State 2: Divider; $z(t-1) = L(r-L\tan\theta)/(S(t-1)+L\tan\theta)$
 ALU; $y_2(t) + y_1(t)$

State 3: Divider; $y'(t) = (y_2(t)-y_1(t))/(y_2(t)+y_1(t))$
 ALU; $x_2(t) - x_1(t)$
 Multiplier; $y(t-1) = z(t-1)*y'(t-1)/L$

State 4: Divider; $y'(t) = (y_2(t)-y_1(t))/(y_2(t)+y_1(t))$
 ALU; $x_2(t) + x_1(t)$
 Multiplier; $x(t-1) = z(t-1)*x'(t-1)/L$

State 5: Divider; $x'(t) = (x_2(t)-x_1(t))/(x_2(t)+x_1(t))$
 Squarer; $y'(t)**2$
 From state 5 to state 11, $x_1(t+1)$, $x_2(t+1)$,
 $y_1(t+1)$, and $y_2(t+1)$ must be loaded.

State 6: Divider; $x'(t) = (x_2(t)-x_1(t))/(x_2(t)+x_1(t))$
 Squarer; $y'(t)**2$

State 7: Divider; $y'(t)/L$
 Multiplier; $x'(t)*x'(t)$

State 8: Divider; $y'(t)/L$
 ALU; $y'(t)**2 + x'(t)*x'(t)$

State 9: Divider; $x'(t)/L$
 Square Root; $S(t) = \text{Sqrt}(y'(t)**2+x'(t)*x'(t))$

State 10: Divider; $x'(t)/L$
 Square Root; $S(t) = \text{Sqrt}(y'(t)**2+x'(t)*x'(t))$

State 11: ALU; $S(t) + L\tan\theta$
 Go to state 1.

4.4 Comparison with Other Processors

The GTP must be compared to the dedicated TP as well as other commercially available general purpose DSPs. This comparison should greatly influence the GTP's marketability. The analysis will be performed at the architecture level since the fastest technology can be used

in any of the different processors.

In order to process the triangulation formulae, the dedicated TP required 7 clock cycles compared to 11 cycles for the GTP. Of course, the cycle of the dedicated TP was based on the divide time, where as the GTP's cycle is based on one half the divide time. Therefore, if one defines one half the divide time as a standard cycle, the dedicated TP requires 14 cycles. Thus, architecturally the GTP performs better than the dedicated architecture. The three cycle difference is easily explainable. Two standard cycles (one dedicated architecture cycle) was lost due to a bidirectional bus transfer. Given a technology which can incorporate the full dedicated TP on a single chip, the bidirectional bus could be eliminated. The final lost cycle is due to a lack of resolution of the dedicated TP's system clock. Upon the final clock cycle where a multiplication takes place, the data is not available until the end of the cycle (divide time) eventhough the multiplication only requires approximately one half of the cycle. With the GTP, the cycle is based on one half the divide time, and the multiplication result is available immediately after processing.

Comparing the GTP to a single chip general purpose DSP results in a resounding speed advantage for the GTP. A typical DSP (TI, Motorola, AD, NEC, Fairchild, etc.) has

a multiply/accumulate unit as well as an ALU. Usually, these two units share the same output bus to RAM and thus, a bottleneck may occur if true parallel operation is required. Assume that parallel operation of the two units can be obtained. Due to the high accuracy (16 to 24 bits) of the DSPs, a typical cycle time of 100ns is obtainable. The GPTP with an 8 to 12 bit accuracy would have a slightly smaller cycle time. Giving the benefit of doubt to the DSPs, assume that the cycle times are equal in length. In two cycles, a DSP can perform two multiply/accumulates and two ALU operations assuming parallel operation or one division and two ALU operations. During the same two cycles, the GPTP can perform 1 division, 2 multiplications, 1 square or square root operation, and 2 ALU operations. Thus, the GPTP has far greater processing capabilities compared with general purpose DSPs. To process the triangulation formulae, a DSP requires at least 15 cycles and that, if only 2 cycles are required for the square root operation (unlikely) and parallel computation occurs between the multiply/accumulate unit and the ALU. Of course, these added capabilities of the GPTP are at the expense of accuracy. One possible enhancement to the GPTP would be to replace the multiplier unit with a multiply/accumulate unit. This may increase its worth in some DSP applications but the added silicon area may be

difficult to provide. This is purely a speed or throughput comparison. Most DSPs are aimed at a different class of applications than the GTP due to their increased accuracy, larger internal RAM, and greater facilities for accessing external RAM if required. Thus, although the GTP is a much faster processor, unless increased accuracy and more data and program RAM is made available, the GTP will never replace other DSPs in certain applications.

4.5 Enhancements and Improvements.

The major areas of interest in improving the GTP is increasing its accuracy and providing more program and data RAM. Due to current technology and silicon area limitations, increased accuracy is only obtainable if some of the other supporting circuitry is reduced. Of course an increase in accuracy would result in larger buses and registers as well as larger processing units. It is therefore evident that increased accuracy, and increased on chip program and data RAM cannot all be accommodated. In fact, even by maintaining current accuracy (8-bits), large improvements in data or program space are not feasible on chip. Thus, external memory support would be required to increase the program and data storage.

4.5.1 Program Memory

By examining the GTPs architecture (figure 4.1), adding external program memory is not difficult. The PC which would be further expanded can be used to address the external memory, and a 32-bit data bus would be required. The current scheme for memory access requires two accesses per system cycle (i.e., instruction broken into two segments). Given even a cycle time of 30ns, modern SRAMs can satisfy this requirement since 13ns CMOS SRAMs are available [7]. It is also possible to decode the entire instruction at the same time. Thus, a 13ns cycle time could be accommodated. A switch could be used to multiplex the desired instruction segment using a pipeline of depth two, the entire next instruction can be accessed while the current one is executing (currently, the next segment is being accessed while the current segment is executing). Conditional jump operations may now cause a half cycle overhead. Other modern techniques such as a small on chip instruction caches (ADSP 210Q), DMA transfer (TI), or a deeper pipeline (Motorola DSP 5000) could be employed to ensure that the program RAM can satisfy the system cycle speed requirements. By allowing external program RAM access, some of the current silicon area reserved for the instruction cache would be eliminated. This additional area could be used to increase accuracy or increase the

size of the data memory.

4.5.2 Data Memory

By referring back to the characteristics of the data memory and decoders it becomes quite evident that no commercially available product could be used to replace this memory. The reason for this is that four units accessing (reads or writes) the data memory in parallel (at the same time) is required. Therefore, the current on chip data memory could be increased in size, or a new external memory chip could be developed. If one of the GPTP's target market area includes the replacement of some DSP's, then more data memory will be required than that which can be incorporated on the chip. Thus, an external RAM chip will be developed. The internal data RAM can be removed resulting in sufficient silicon area to encompass 16-bits of accuracy. The main problem is the design of such a data memory in sufficient size to justify its existence.

Another justification for the existence of this multi-accessible RAM would be its usefulness in other applications. Certainly, for many multiprocessor applications (after all, this is what the GPTP essentially is), a shared global memory which can be successfully accessed in parallel would result in a reduction in communication overhead. Currently, a particular node of a

multiprocessor system must spend much of its time communicating with the other nodes. This communication is done by the applications processor itself or by a dedicated communications processor at each node (B-Hive Multicomputer [2]). Either way, there is an associated communication overhead and cost. By using a global shared memory, the communications could be greatly reduced since each processor would have no need of requesting and waiting for the data from another processor. Thus, a global shared memory, if it can truly operate in parallel, would have many applications in the multiprocessor field [3][8][13].

4.5.2.1 Requirements and Constraints for Data Memory

There are several requirements and constraints which must be adhered to in the implementation of the shared global memory. Each unit linking to the memory must have the ability to access the memory in parallel. Of course, this forces the constraint that at most one unit can be writing to a particular memory location at any one time. Thus, some form of wait signal or queue must be implemented. In order to reduce a potential backlog of read requests, all reads will be performed immediately regardless of other units reading or writing. This will result in a potential multiple read of a particular memory location at the same time. In order to remain compatible

with such multiprocessor architectures as the B-Hive, the memory must have asynchronous access. The B-Hive computer performs asynchronous communication between the nodes and there is no system clock. In this way each node can process at its own rate and no system clock is required. The final requirement is to determine a suitable number of accessing units. Four was somewhat arbitrarily chosen. Certainly the number would be somewhat dependant on the application as well as the fabrication process. Four was chosen to be compatible with the General Purpose Triangulation Processor shared data memory. Certainly, the design could easily be expanded to encompass the desired number of accessing units. The only real design constraint is NT's 3um CMOS double metal process. As this is not a "RAM technology", the packing density and speed will not be as good as that obtainable in industry. The goal is more to prove the functionality and feasibility of the shared memory rather than designing an industry standard product.

4.5.2.2 Memory Chip Architecture

The memory chip architecture is roughly based on the structure of the MM2114 Static RAM [19]. The proposed architecture is shown in figure 4.3. The main difference between the multi-access RAM structure and that of the 2114 is the collision detection circuitry and the added row and

Figure 4.3 : Multi-Access RAM Architecture

column decoders. The address transition detector (ATD) and the stable address transition circuit (SAT) provide asynchronous operation. Basically, the memory is accessed in the following manner. For a particular unit to read memory, the associated write signal (WR) must remain low. The collision detector circuitry does not check for a collision if a unit is not writing. The unit then sends its address to the address bus. The ATD and SAT circuits detect a change of address. This forces an immediate precharge of the particular unit's bit lines as the address is being decoded in the row and column decoders. Once the address has been decoded and the precharge is completed, the correct memory location can be accessed. In order to write to a memory location, the WR signal must be high. The collision detection units will then determine if there are any other units writing or waiting to write to that particular location. If there is no collision, the address is decoded, and the memory cell written. If there is a collision, then the requesting unit is sent a wait signal. As soon as the location becomes free, the wait signal is removed. Write access priority is built in so that for requests arriving simultaneously or for several units waiting, the same unit always gets first access. This may appear as a possible starvation situation but since a unit can remove its write request (make WR low), a unit can

avoid starvation.

4.5.2.3 Multi-Write Detection

A multi-write occurs when two or more units attempt to write to a particular memory location at the same time. Figure 4.4 shows the circuit for detecting such a collision between units A & B. The associated RAMCOLLISION HILO simulation plot (figure 4.5) is also given to show its functionality. For a four unit shared memory, six multi-write detectors are required. The outputs of these will be denoted as CD12, CD13, CD14, CD23, CD24, CD34 (meaning collision between units 1 & 2, etc.). Unit 1 has the highest priority down to unit 4. This is encoded by generating four multi-write detection signals, CD1, CD2, CD3, and CD4. These signals are produced as follows:

$$CD1 = CD12 * \overline{WAIT2} + CD13 * \overline{WAIT3} + CD14 * \overline{WAIT4}$$

$$CD2 = CD12 + CD23 * \overline{WAIT3} + CD24 * \overline{WAIT4}$$

$$CD3 = CD13 + CD23 + CD34 * \overline{WAIT4}$$

$$CD4 = CD14 + CD24 + CD34$$

Taking CD2 as a typical example; if a multi-write occurs between units 1 & 2, then unit 2 always waits; if a multi-write occurs between units 2 & 3 or 2 & 4 and units three or four are not waiting then unit 2 waits. This means that unit 1 has higher priority than unit 2 and unit

Figure 4.4 : Multi-Write Detector

Figure 4.5 : Multi-Write Detector

2 only waits on unit 3 and unit 4 if these units are already in the middle of their write operation. If units 3 & 4 are both waiting, then unit 2 has higher priority. Once a multi-write has been detected, the appropriate wait signal is generated. The waiting unit also has its column selection disabled.

4.5.2.4 Other Circuitry

In order to provide asynchronous memory access, a change in address must be detected. Also, since each bit of an address may arrive at slightly different times due to varying delays, a stable address must be detected [15]. The Address Transition Detector (ATD) is shown in figure 4.6 and the Stable Address Transition (SAT) circuit [7] is shown in figure 4.7. Their associated HILO functional simulations are shown in the RAMATD and RAMSAT plots (figures 4.8 and 4.9 respectively). When an address bit changes, a pulse is outputted from the ATD circuit. The SAT circuit detects the pulses and waits until all pulses finish. The SAT signal along with a read request causes the generation of the bit line precharge signal. This precharge signal causes the bit lines to be precharged to 5 volts while disconnecting the appropriate unit's row and column decoders. For the example architecture, 8 bit addresses were used, therefore 32 ATD circuits and 4 SAT

Figure 4.8 : ATD

Figure 4.9 : SAT

circuits are required.

The row and column decoders are identical since the 8-bit address is partitioned into 16 rows and 16 columns. The decoder circuit is shown in figure 4.10. This is a simple AND gate 4 bit decoder. Four column and four row decoders are required.

The sense amplifier is a differential amplifier with a three stage output. Four such units are required. Due to the drive capabilities of the designed memory cell (shown later), the sense amplifier will not have any difficult constraints and thus, most amplifiers will perform adequately. If a larger amount of memory is manufactured on a larger pad frame or a shorter channel process, then more advanced techniques for sense amplification may be needed.

4.5.2.5 Memory Cell Design

The four unit memory cell is made up of 12 transistors (2 PMOS, 10 NMOS). The schematic is shown in figure 4.11 [25]. Four bit and four bitbar lines are needed per cell as well as four word lines. A transistor ratio between the pulldown: wordline: pullup was derived to be 3:2:1. This was simulated using SPICE. The various nodes can be seen in figure 4.11. The initial simulation used a 2pF bit line, but the area of sources and drains were not considered.

Figure 4.10 : Row and Column Decoder Circuit

Figure 4.11 : RAM Cell Circuit

The plot from the files ramcell & ramcellwav show the simulation results (figures 4.12 and 4.13). Initially, a logic 1 is successfully written into the cell; Units 2 and 3 then try to read the cell at the same time; and finally, unit 1 writes a 0 while unit 4 reads the cell. The write time is quite sufficient, but the fall time is quite long. This suggests a larger pulldown so that the precharged bit lines can discharge faster.

The layout of the memory cell is shown in figure 4.14. The shared drain wordline pass transistors take up the right hand side of the figure, whereas the flip-flop occupies the left hand side. Due to the shared drain structure, more room than anticipated was provided for the flip-flop. Thus increased pulldown and pullup transistors were used. The total cell size is approximately 96um by 120um for a 3um process. The cells have been designed to butt directly together. Also by flipping the cells, two rows can share a single ground rail and a single power rail. The bit lines run horizontally in metal2 and the word lines run vertically in polysilicon and metall. Simulation for the layout was performed using SPICE. The plot for file ramcell1 (figure 4.15) shows the actual layout simulation with 2pF bit lines. The input waveforms are identical to the prelayout input signals. The discharge through the pulldown has been greatly improved.

Figure 4.12 : Pre-layout RAM Cell

Figure 4.13 : Input Waveforms

Figure 4.14 : Multi-Access RAM Cell Layout

Figure 4.15 : Post-layout RAM Cell

The write operation takes longer than anticipated, and thus, the pullup transistors may have to be increased in size. Of course, in terms of system timing, the write operation at the memory cell level can be slower than the read, since the read cell also has the added precharge time which is not shown here. The plot from file ramcell2 (figure 4.16) shows unit 1 writing a logic 0 while the other three units all try to read the same location. The rise/fall time from this last plot appears to be around 25ns (quite good considering the process).

4.5.2.6. Enhancements and Conclusions

There are many possible enhancements or improvements possible for the shared memory chip. First, a smaller more appropriate RAM technology could be used to increase speed [20] and packing density. A request queue could be implemented [11] so that a particular unit would not have to wait (can be currently done by the unit itself). The priority could be programmable or perhaps first come first served. The best improvement would be to generate a finish signal, since depending on the number of units accessing the cell, the access time differs. Various more advanced circuitry for sense amplifiers and power down circuitry could be employed given the silicon area [1][5][15][16][21][22][23].

Overall it appears that a multi-unit shared memory is

Figure 4.16 : Post-layout RAM-Cell 2

feasible and quite implementable. Due to the multiprocessor need for such global memory, there are increasing trends toward this direction. The development of dual-port RAM to LSI Logic Corporation's three and five port RAM show industries involvement in this area.

4.6 Applications

There are many suitable applications for the GTP. Initially, it was targetted at low resolution sensor applications. These applications generally require high processing power with a relatively small data set. A single chip GTP was designed which included on chip program and data memory. Due to the vast processing capabilities of the GTP, it could be targetted at the general purpose DSP market. In order to become competitive, the program and data memory had to be expanded while the accuracy was increased. Thus, program and data store were taken off chip which left room for a 16-bit accuracy. The processing power of the new GTP is greater than typical DSPs and thus, suitable for most DSP applications. The specialized data RAM used for the new enhanced GTP can also be used extensively for other multiprocessor applications. Two versions of the GTP seem to be feasible and are targetted at different classes of

applications, sensor and DSP applications. The results are summarized in Table 4.1.

4.7 Summary

A general purpose triangulation processor has been designed following stringent criteria and constraints. The vast processing power provided by the GTP on a single chip, coupled with a user friendly instruction set, and basic communication access, makes the GTP a good choice for scientific sensor applications. Little programming knowledge is required. In comparison to other DSPs and to the dedicated TP, the GTP performed faster. Once the GTP was enhanced with greater accuracy, and larger amounts of program and data storage, it seems to out distance most DSPs in performance, but may cost more due to the extra memory chips. Currently, large research interests in algorithms to detect parallelism is in progress. Such algorithms could be useful when programming the GTP. The multi-user shared data memory was designed and should prove exceptionally useful for multiprocessor applications. In particular, the GTP and the multi-user RAM are probably more cost effective than the dedicated TP or other solutions due to their wide range of applications. Basically any application requiring fast mathematical computation can make use of the GTP.

Comparison of the Two Forms of the GTP

	<u>Basic GTP</u>	<u>Enhanced GTP</u>
Applications	Sensors	DSP
Internal Data RAM	32 by 8-bits	None
External Data RAM	None	Yes
Internal Program RAM	32 by 64-bits	Possible cache
External Program RAM	None	Yes
Accuracy	8 to 12 bits	16 bits
Multiplier Type	Multiplier	Multiply/Accumulator

Table 4.1 : Comparison of GTPs

CHAPTER 5: CONCLUSION.

5.0 Summary

This thesis has examined the point method of machine vision with its associated triangulation formulae. In particular, parallel and multiprocessing techniques have been used and further developed in achieving high performance throughput for relatively low resolution sensor applications.

For strict research use with the vision system presented, the dedicated processor provides sufficient processing speed while still conforming to processing constraints typical at the University level. This dedicated solution was designed and developed.

For increased flexibility, the general purpose processor could be developed using a 1.5 micron CMOS process. Such a processor could be used in many sensor applications including the proposed vision system. The development of multi-access global RAM resulted in a powerful tool for multiprocessor systems. With the use of this global memory coupled with the general purpose processor, an extremely powerful DSP can be achieved. The RAM itself was fully simulated but lacked the technology for proper implementation.

5.1 Recommendations

This thesis furthered current research in three main areas. The vision system coupled with the VLSI dedicated processor has improved the throughput of such a point method system. The University of Ottawa is planning on continued testing and development of the dedicated TP which should finally result in the development of the entire vision system.

The development of the general purpose processor results in some interesting concepts with regard to parallel processing and single chip multiprocessing. Currently, there are no plans for implementation of the GTPP due to its processing and silicon requirements. It is likely that this processor will never be further developed although its concepts should appear in some future generation DSPs.

The development of shared global RAM appears to provide advantages in multiprocessor systems. Industry is currently expanding in this direction and university research will likely continue in more depth on this subject.

REFERENCES

1. Aadsen, D., et. al., "Automation of Bist for Embedded RAM", IEEE 1987 Custom Integrated Circuits Conference, pp. 66-67.
2. Agrawal, D., et. al., "B-HIVE Alpha Structure Multi-Computer", Second Conference on Hypercube Multiprocessors, pp. 1-7.
3. Agrawal, D., et. al., "Evaluating the Performance of Multicomputer Configurations", IEEE Computer, May 1986, pp. 23-37.
4. Aleksander, I., "Artificial Vision for Robots", Chapman & Hall, 1983, pp. 9-42, 179-233.
5. Chan, T., et. al., "Advanced Structured Arrays Combine High Density Memories with Channel-Free Logic Array", IEEE 1987 Custom Integrated Conference, pp. 39-43.
6. Cook, C., Morris, T., et. al., "Development of a VLSI Vision Processor using the CHIPSMITH Design Environment", 1987 CCVLSI Conference, pp. 263-268.
7. Flannagan, S., et. al., "Two 13-ns 64K CMOS SRAMs with Very Low Active Power and Improved Asynchronous Circuit Techniques", IEEE Journal of Solid-State Circuits, Vol. sc-21, No. 5, Oct. 1986, pp. 692-702.
8. Goodman, J., et. al., "Hypertree: A Multiprocessor Interconnection Topology", IEEE Transactions on Computers, Vol. c-30, No. 12, Dec. 1981, pp. 923-933.
9. Hamacher, C., et. al., "Computer Organization", McGraw-Hill Inc., 1984, pp. 264-268.
10. Hammamatsu Corp., "Position Sensitive Detectors", 1985, pp. 1-14.
11. Hashimoto, M., et. al., "A 20ns 256K X 4 FIFO Memory", IEEE 1987 Custom Integrated Circuits Conference, pp. 315-318.
12. Hwang, K., "Computer Arithmetic", John Wiley & Sons, 1979, pp. 69-281.
13. Hwang, K., Briggs, F., "Parallel Computer Architecture", McGraw-Hill Inc., 1984.

14. Kanade, T., Sommer, T., "An Optical Proximity Sensor for Measuring Surface Position and Orientation for Robot Manipulation", The Vision Laboratory, Carnegie-Mellon Univ., 1983, pp. 1-26.
15. Kayano, S., et. al., "25-ns 256K X 1/64K X 4 CMOS SRAMs", IEEE Journal of Solid-State Circuits, Vol. sc-21, No. 5, Oct. 1986, pp. 686-690.
16. Kobayashi, Y., et. al., "A 10uW Standby Power 256K CMOS SRAM", IEEE Journal of Solid-State Circuits, Vol. sc-20, No. 5, Oct. 1985, pp. 935-940.
17. Langdon, G., "Computer Design", Computeach Press Inc., 1982, pp. 186-271, 549-550.
18. Morris, T., Chan, C., "VLSI Solution to Triangulation Formulae", 1987 UGIM Conference, pp. 200-205.
19. National Semiconductor, "MM2114 Data Sheets".
20. Nishio, Y., et. al., "0.24ns 7K Hi-BiCMOS Gate Array with Configurable 3-Port 4.6K SRAM", IEEE 1987 Custom Integrated Circuits Conference, pp. 203-204.
21. Nogami, K., et. al., "1-Mbit Virtually Static RAM", IEEE Journal of Solid-State Circuits, Vol. sc-21, No. 5, Oct. 1986, pp. 662-667.
22. Schuster, S., et. al., "A 15-ns CMOS 64K RAM", IEEE Journal of Solid-State Circuits, Vol. sc-21, No. 5, Oct. 1986, pp. 704-711.
23. Sood, L., et. al., "A Fast 8K X 8 CMOS SRAM with Internal Power Down Techniques", IEEE Journal of Solid-State Circuits, Vol. sc-20, No. 5, Oct. 1985, pp. 941-949.
24. Vaillancourt, S., Krieger, M., "Segmented Microprogramming", Univ. of Ottawa.
25. Weste, N., Eshraghian, K., "CMOS VLSI Design", Addison-Wesley, 1985, pp. 348-364.
26. Lepage, P., Univ. of Ottawa Co-op Report, 1985.
27. Rioux, M. et. al., "Design of a Large Depth View Three Dimensional Camera for Robot Vision", Optical Engineering Vol. 26, No. 12, Dec. 1987, pp. 1245-1250.

APPENDIX A : SPLICE MODELS

```

%DEFINE ELMOD1
%NUMOUT 1
%NUMIN 2
%NUMPAR 0
%DECLARE
%
%   REAL CON
%   DATA CON/0/
%   INTEGER OUTFLAG, INFLAG
%   DATA OUTFLAG/0/, INFLAG/0/
%BEGIN
%   GENFLG = .TRUE.
%   IF (FRSTIM .EQV. .TRUE.) THEN
%       CALL CONTROL1(CON, OUTFLAG, INFLAG, VIN1, VIN2)
%       VOUT1=CON
%   ENDIF
%SUBRT
%   SUBROUTINE CONTROL1(CONT, OUT, IN, POR, START)
%       REAL CONT, POR, START
%       INTEGER OUT, IN, KNT, FLAG
%       DATA KNT/3/, FLAG/0/
%       IF (POR .GT. 4.) THEN
%           CONT=7.
%           FLAG=1
%       ELSE
%           IF ((CONT .EQ. 7.) .AND. (FLAG .EQ. 1)) THEN
%               IF (START .EQ. 5.) THEN
%                   CONT=1.
%                   IN=1
%                   OUT=0
%                   KNT=3
%                   FLAG=0
%               ENDIF
%           ELSE
%               IF ((KNT .EQ. 0) .AND. (CONT .NE. 7.)) THEN
%                   IN=1
%                   OUT=0
%                   KNT=3
%                   CONT=CONT+1
%               ELSE
%                   IF (KNT .EQ. 0) THEN
%                       FLAG=1
%                   ENDIF
%                   KNT=KNT-1
%                   IN=0
%                   OUT=0
%                   IF (KNT .EQ. 0) THEN
%                       OUT=1
%                   ENDIF
%               ENDIF
%           ENDIF
%       ENDIF
%       RETURN
%   END
%ENDSUBRT
%END
%DEFINE ELMOD2
%NUMOUT 1
%NUMIN 7
%NUMPAR 3
%DECLARE
%   REAL X1, X2, Y1, Y2, YP, XP, S, SUBX, SUBY, ADDX, ADDY, N, D, CON

```

```

% DATA X1/0/,X2/0/,Y1/0/,Y2/0/,YP/0/,XP/0/,S/0/,N/0/,D/0/
% DATA SUBX/0/,SUBY/0/,ADDX/0/,ADDY/0/,CON/0/
% INTEGER INKNT,YFLAG,XFLAG,OUTFLAG,INFLAG
% DATA INKNT/5/,YFLAG/0/,XFLAG/0/,OUTFLAG/0/,INFLAG/0/
%BEGIN
% GENFLG=.TRUE.
% IF (FRSTIM .EQV. .TRUE.) THEN
%   CALL CONTROL2(CON,OUTFLAG,INFLAG,VIN5,VIN6)
%   IF (OUTFLAG .EQ. 1) THEN
%     IF (D .NE. 0) THEN
%       VOUT1=N/D
%     ELSE
%       WRITE(9,*)'DIVIDING BY 0 AT TIME: ',TIME
%       VOLD=-1.
%     ENDIF
%   ENDIF
%   IF (CON .EQ. PAR(3)) THEN
%     WRITE(9,*)'GETTING INPUTS AT TIME: ',TIME
%     X1=VIN1
%     X2=VIN2
%     Y1=VIN3
%     Y2=VIN4
%     INKNT=4
%   ENDIF
%   IF (INKNT .EQ. 0) THEN
%     SUBX=X2-X1
%     SUBY=Y2-Y1
%     ADDX=X2+X1
%     ADDY=Y2+Y1
%     INKNT=5
%     WRITE(9,*)'ADD & SUB FINISHED AT TIME: ',TIME
%   ENDIF
%   IF (INKNT .NE. 5) THEN
%     INKNT=INKNT-1
%   ENDIF
%   IF (CON .EQ. 1.) THEN
%     IF (INFLAG .EQ. 1) THEN
%       N=SUBY
%       D=ADDY
%       YFLAG=1
%       WRITE(9,*)'(Y2-Y1)/(Y2+Y1) AT TIME: ',TIME
%     ENDIF
%   ENDIF
%   IF (CON .EQ. 2.) THEN
%     IF (INFLAG .EQ. 1) THEN
%       N=SUBX
%       D=ADDX
%       XFLAG=1
%       WRITE(9,*)'(X2-X1)/(X2+X1) AT TIME: ',TIME
%       IF (YFLAG .EQ. 1) THEN
%         YP=VOUT1
%         YFLAG=0
%       ENDIF
%     ENDIF
%   ENDIF
%   IF (CON .EQ. 3.) THEN
%     IF (INFLAG .EQ. 1) THEN
%       N=YP
%       D=PAR(1)
%       WRITE(9,*)'YP/L AT TIME: ',TIME
%       IF (XFLAG .EQ. 1) THEN
%         XP=VOUT1

```

```

      XFLAG=0
      ENDIF
    ENDIF
  ENDIF
  IF (CON .EQ. 4.) THEN
    IF (INFLAG .EQ. 1) THEN
      N=XP
      D=PAR(1)
      WRITE(9,*)'XP/L AT TIME: ',TIME
    ENDIF
  ENDIF
  IF (CON .EQ. 5.) THEN
    IF (INFLAG .EQ. 1) THEN
      S=VIN7
    ENDIF
  ENDIF
  IF (CON .EQ. 6.) THEN
    IF (INFLAG .EQ. 1) THEN
      N=PAR(2)
      D=S
      WRITE(9,*)'L(R-LTAN)/S AT TIME: ',TIME
    ENDIF
  ENDIF
ENDIF
%SUBRT
      SUBROUTINE CONTROL2(CONT,OUT,IN,POR,START)
      REAL CONT,POR,START
      INTEGER OUT,IN,KNT,FLAG
      DATA KNT/3/,FLAG/0/
      IF (POR .GT. 4.) THEN
        CONT=7.
        FLAG=1
      ELSE
        IF ((CONT .EQ. 7.) .AND. (FLAG .EQ. 1)) THEN
          IF (START .EQ. 5.) THEN
            CONT=1.
            IN=1
            OUT=0
            KNT=3
            FLAG=0
          ENDIF
        ELSE
          IF ((KNT .EQ. 0) .AND. (CONT .NE. 7.)) THEN
            IN=1
            OUT=0
            KNT=3
            CONT=CONT+1
          ELSE
            IF (KNT .EQ. 0) THEN
              FLAG=1
            ENDIF
            KNT=KNT-1
            IN=0
            OUT=0
            IF (KNT .EQ. 0) THEN
              OUT=1
            ENDIF
          ENDIF
        ENDIF
      ENDIF
      RETURN
    END

```

```

%ENDSUBRT
%END
%DEFINE ELMOD3
%NUMOUT 1
%NUMIN 3
%NUMPAR 0
%DECLARE
%   REAL OP1,OP2,Z,XPL,YPL,YP,XP,CON
%   DATA OP1/0/,OP2/0/,Z/0/,XPL/0/,YPL/0/,XP/0/,YP/0/,CON/0/
%   INTEGER OUTFLAG,INFLAG
%   DATA OUTFLAG/0/,INFLAG/0/
%BEGIN
%   GENFLG=.TRUE.
%   IF (FRSTIM .EQV. .TRUE.) THEN
%       CALL CONTROL3(CON,OUTFLAG,INFLAG,VIN2,VIN3)
%       IF (OUTFLAG .EQ. 1) THEN
%           VOUT1=OP1*OP2
%       ENDIF
%       IF (CON .EQ. 1.) THEN
%           IF (INFLAG .EQ. 1) THEN
%               OP1=Z
%               OP2=XPL
%               WRITE(9,*)'MULT Z*(XP/L) AT TIME: ',TIME
%           ENDIF
%       ENDIF
%       IF (CON .EQ. 2.) THEN
%           IF (INFLAG .EQ. 1) THEN
%               YP=VIN1
%               OP1=YP
%               OP2=YP
%               WRITE(9,*)'MULT YP*YP AT TIME: ',TIME
%           ENDIF
%       ENDIF
%       IF (CON .EQ. 3.) THEN
%           IF (INFLAG .EQ. 1) THEN
%               XP=VIN1
%               OP1=XP
%               OP2=XP
%               WRITE(9,*)'MULT XP*XP TIME: ',TIME
%           ENDIF
%       ENDIF
%       IF (CON .EQ. 4.) THEN
%           IF (INFLAG .EQ. 1) THEN
%               YPL=VIN1
%           ENDIF
%       ENDIF
%       IF (CON .EQ. 5.) THEN
%           IF (INFLAG .EQ. 1) THEN
%               XPL=VIN1
%           ENDIF
%       ENDIF
%       IF (CON .EQ. 7.) THEN
%           IF (INFLAG .EQ. 1) THEN
%               Z=VIN1
%               OP1=Z
%               OP2=YPL
%               WRITE(9,*)'MULT Z*(YP/L) AT TIME: ',TIME
%           ENDIF
%       ENDIF
%   ENDIF
%ENDIF
%SUBRT
%   SUBROUTINE CONTROL3(CONT,OUT,IN,POR,START)

```



```

%      REAL CONT,POR,START
%      INTEGER OUT,IN,KNT,FLAG
%      DATA KNT/3/,FLAG/0/
%      IF (POR .GT. 4.) THEN
%          CONT=7.
%          FLAG=1
%      ELSE
%          IF ((CONT .EQ. 7.) .AND. (FLAG .EQ. 1)) THEN
%              IF (START .EQ. 5.) THEN
%                  CONT=1.
%                  IN=1
%                  OUT=0
%                  KNT=3
%                  FLAG=0
%              ENDIF
%          ELSE
%              IF ((KNT .EQ. 0) .AND. (CONT .NE. 7.)) THEN
%                  IN=1
%                  OUT=0
%                  KNT=3
%                  CONT=CONT+1
%              ELSE
%                  IF (KNT .EQ. 0) THEN
%                      FLAG=1
%                  ENDIF
%                  KNT=KNT-1
%                  IN=0
%                  OUT=0
%                  IF (KNT .EQ. 0) THEN
%                      OUT=1
%                  ENDIF
%              ENDIF
%          ENDIF
%      ENDIF
%      RETURN
%      END
%ENDSUBRT
%END
%DEFINE ELMOD4
%NUMOUT 1
%NUMIN 7
%NUMPAR 0
%DECLARE
%      REAL Z,X,Y,CON
%      DATA Z/0/,X/0/,Y/0/,CON/0/
%      INTEGER KNT,OUTFLAG,INFLAG
%      DATA KNT/1/,OUTFLAG/0/,INFLAG/0/
%BEGIN
%      GENFLG=.TRUE.
%      IF (FRSTIM .EQV. .TRUE.) THEN
%          CALL CONTROL4(CON,OUTFLAG,INFLAG,VIN2,VIN6)
%          IF (CON .EQ. 7.) THEN
%              IF (INFLAG .EQ. 1) THEN
%                  Y=VIN1
%              ENDIF
%          ENDIF
%          IF (CON .EQ. 2.) THEN
%              IF (INFLAG .EQ. 1) THEN
%                  X=VIN1
%              ENDIF
%          ENDIF
%          IF (CON .EQ. 7.) THEN

```

```

%           IF (INFLAG .EQ. 1) THEN
%               Z=VIN7
%           ENDIF
%       ENDIF
%       IF (VIN3 .EQ. 5.) THEN
%           VOUT1=Z
%       ENDIF
%       IF (VIN4 .EQ. 5.) THEN
%           VOUT1=Y
%       ENDIF
%       IF (VIN5 .EQ. 5.) THEN
%           VOUT1=X
%       ENDIF
%   ENDIF
%SUBRT
%       SUBROUTINE CONTROL4(CONT,OUT,IN,POR,START)
%       REAL CONT,POR,START
%       INTEGER OUT,IN,KNT,FLAG
%       DATA KNT/3/,FLAG/0/
%       IF (POR .GT. 4.) THEN
%           CONT=7.
%           FLAG=1
%       ELSE
%           IF ((CONT .EQ. 7.) .AND. (FLAG .EQ. 1)) THEN
%               IN=0
%               IF (START .EQ. 5.) THEN
%                   CONT=1.
%                   IN=1
%                   OUT=0
%                   KNT=3
%                   FLAG=0
%               ENDIF
%           ELSE
%               IF ((KNT .EQ. 0) .AND. (CONT .NE. 7.)) THEN
%                   IN=1
%                   OUT=0
%                   KNT=3
%                   CONT=CONT+1
%               ELSE
%                   IN=0
%                   IF (KNT .EQ. 0) THEN
%                       IN=1
%                       FLAG=1
%                   ENDIF
%                   KNT=KNT-1
%                   OUT=0
%                   IF (KNT .EQ. 0) THEN
%                       OUT=1
%                   ENDIF
%               ENDIF
%           ENDIF
%       ENDIF
%       RETURN
%   END
%ENDSUBRT
%END
%DEFINE ELMOD5
%NUMOUT 1
%NUMIN 3
%NUMPAR 1
%DECLARE
%       REAL YP2,XP2,CON

```

```

%      DATA YP2/0/,XP2/0/,CON/0/
%      INTEGER OUTFLAG,INFLAG
%      DATA OUTFLAG/0/,INFLAG/0/
%BEGIN
%      GENFLG=.TRUE.
%      IF (FRSTIM .EQV. .TRUE.) THEN
%          CALL CONTROL5(CON,OUTFLAG,INFLAG,VIN2,VIN3)
%          IF ((CON .EQ. 3.) .AND. (INFLAG .EQ. 1)) THEN
%              YP2=VIN1
%          ENDIF
%          IF ((CON .EQ. 4.) .AND. (INFLAG .EQ. 1)) THEN
%              XP2=VIN1
%          ENDIF
%          IF ((CON .EQ. 4.) .AND. (OUTFLAG .EQ. 1)) THEN
%              VOUT1=PAR(1)+SQRT(XP2+YP2)
%              WRITE(9,*)'ADD-SQRT-ADD AT TIME: ',TIME
%          ENDIF
%      ENDIF
%SUBRT
%      SUBROUTINE CONTROL5(CONT,OUT,IN,POR,START)
%      REAL CONT,POR,START
%      INTEGER OUT,IN,KNT,FLAG
%      DATA KNT/3/,FLAG/0/
%      IF (POR .GT. 4.) THEN
%          CONT=7.
%          FLAG=1
%      ELSE
%          IF ((CONT .EQ. 7.) .AND. (FLAG .EQ. 1)) THEN
%              IF (START .EQ. 5.) THEN
%                  CONT=1.
%                  IN=1
%                  OUT=0
%                  KNT=3
%                  FLAG=0
%              ENDIF
%          ELSE
%              IF ((KNT .EQ. 0) .AND. (CONT .NE. 7.)) THEN
%                  IN=1
%                  OUT=0
%                  KNT=3
%                  CONT=CONT+1
%              ELSE
%                  IF (KNT .EQ. 0) THEN
%                      FLAG=1
%                  ENDIF
%                  KNT=KNT-1
%                  IN=0
%                  OUT=0
%                  IF (KNT .EQ. 0) THEN
%                      OUT=1
%                  ENDIF
%              ENDIF
%          ENDIF
%      ENDIF
%      RETURN
%      END
%ENDSUBRT
%END
SPICE
SYSTEM
.MODEL IPOR TSRC:0 5 0 200US 0US 4US 5US 8US 9US 200US
.MODEL ISTCON TSRC:0 5 0 52US 0US 16US 17US 28US 29US 52US

```

```

.MODEL IIN1 TSRC:2 16 0 200US 0US 28US 29US 56US 57US 200US
.MODEL IIN2 TSRC:17 58 0 200US 0US 28US 29US 56US 57US 200US
.MODEL IIN3 TSRC:14 125 0 200US 0US 28US 29US 56US 57US 200US
.MODEL IIN4 TSRC:5 10 0 200US 0US 28US 29US 56US 57US 200US
.MODEL ISELZ TSRC:0 5 0 200US 0US 99US 100US 105US 106US 150US $
                                151US 155US 156US
.MODEL ISELY TSRC:0 5 0 200US 0US 107US 108US 112US 113US 157US $
                                158US 164US 165US
.MODEL ISELX TSRC:0 5 0 200US 0US 127US 128US 132US 133US 180US $
                                181US 185US 186US

.MODEL CONTROL ELMOD1:NUMOUT=1 NUMIN=2
.MODEL INDIV ELMOD2:NUMOUT=1 NUMIN=7 PAR1=10 PAR2=163.6 PAR3=5
.MODEL MULTI ELMOD3:NUMOUT=1 NUMIN=3
.MODEL TESTO ELMOD4:NUMOUT=1 NUMIN=7
.MODEL ADDSQRT ELMOD5:NUMOUT=1 NUMIN=3 PAR1=3.64
DIV DIVBUS IN1 IN2 IN3 IN4 POR STCON S INDIV
MULT MULTBUS DIVBUS POR STCON MULTI
ASQRTA S MULTBUS POR STCON ADDSQRT
PORI POR IPOR
OUTP OUTBUS MULTBUS POR SELZ SELY SELX STCON DIVBUS TESTO
CON CONBUS POR STCON CONTROL
STCONI STCON ISTCON
TESTI SELZ ISELZ
IN1I IN1 IIN1
IN2I IN2 IIN2
IN3I IN3 IIN3
IN4I IN4 IIN4
SELYI SELY ISELY
SELXI SELX ISELX
.TIME 1US 200US
.PRINT POR CONBUS STCON
.PRINT IN1 IN2 IN3 IN4 CONBUS
.PRINT DIVBUS MULTBUS S CONBUS
.PRINT OUTBUS SELZ SELY SELX CONBUS
GO
END

```

APPENDIX B : CHIPSMITH MODEL

```
{ VISION PROCESSOR
{ WRITTEN BY CLIVE COOK AND TODD MORRIS
{ AUGUST 1987
```

```
{ This implementation uses cells which can be routed in metal two
and metal one
```

```
{ only. To route in metal one and poly you must change the cells.
```

```
{ All cells including pad cells are for double metal routing
```

```
{ ----- }
```

```
Constant bits = 8
```

```
EXTERNALSIGNAL div(1:8), out(1:8), out1(1:8), div1(1:8)
```

```
EXTERNALSIGNAL bus2(1:8), bus3(1:8), sb
```

```
EXTERNALSIGNAL outlbus(1:8), divbus(1:8)
```

```
{ ----- }
```

```
{ Output and Input Pad definitions are macros which also give
```

```
{ access to the outside world
```

```
Input Pad (cinp02) consel, outsel1, outsel2, conload, sx, sy
```

```
Input Pad (cinp02) conbus(1:8)
```

```
{ ----- }
```

```
Pad (ciop02) iopad1
```

```
Pad (ciop02) iopad2
```

```
Pad (ciop02) iopad3
```

```
Pad (ciop02) iopad4
```

```
Pad (ciop02) iopad5
```

```
Pad (ciop02) iopad6
```

```
Pad (ciop02) iopad7
```

```
Pad (ciop02) iopad8
```

```
Pad (ciop02) iopad9
```

```
Pad (ciop02) iopad10
```

```
Pad (ciop02) iopad11
```

```
Pad (ciop02) iopad12
```

```
Pad (ciop02) iopad13
```

```
Pad (ciop02) iopad14
```

```
Pad (ciop02) iopad15
```

```
Pad (ciop02) iopad16
```

```
{ ----- }
```

```
Pad (cgnd01) vsspad
```

```
Pad (cvdd01) vddpad
```

```
vddpad; vsspad
```

```
{ ----- }
```

```
{ GLOBAL SIGNALS
```

```
Signal addbus(1:bits)
```

```
Signal al6(1:2*bits), en1, add16(1:2*bits)
```

```
Signal xy(1:bits), multbus(1:2*bits)
```

```
Signal a8(1:bits)
```

```
Signal m1(1:bits), m2(1:bits)
```

```
Signal sqroot(1:bits), sx1, syl
```

```

Signal divlbus(1:8), outbus(1:8)

( ----- )

Part halfadder[a,b] -> sum,cout
  minv01[mnan02[a,b]] -> cout
  mnor02[mnor02[a,b],cout] -> sum
End

( ----- )

Part fulladder[a,b,cin] -> sum, cout

  Signal tmpsum, tmpcout1, tmpcout2

  halfadder[a,b] -> tmpsum, tmpcout1
  halfadder[tmpsum,cin] -> sum, tmpcout2
  minv01[mnor02[tmpcout2,tmpcout1]] -> cout
End

( ----- )

Part mult[plicand,plier,ipp,cin] -> opp,cout
  fulladder[ipp,minv01[mnan02[plicand,plier]],cin] -> opp,cout
End

( ----- )

Part varmult(bits) [multiplicand(0:bits-1),multiplier(0:bits-1)] ->
  result(0:2*bits-1)

  Integer j, i

  Signal ppin, carryin, nextpp(0:bits-1)
  Signal tcout(0:bits*bits-1)
  Signal toutpp(0:bits*bits-1)

  GND -> ppin
  GND -> carryin
  FOR i = 0:bits-1 CYCLE
    FOR j = 0:bits-1 CYCLE
      IF (i=0) AND (j=0) THEN
        mult[multiplicand(0),multiplier(0),ppin,carryin] ->
          result(0), tcout(0)
      ELSE IF i = 0 THEN
        mult[multiplicand(j),multiplier(0),ppin,tcout(j-1)] ->
          toutpp(j), tcout(j)
        IF j = bits-1 THEN tcout(bits-1) -> nextpp(0)
      ENDIF
    ELSE IF j = 0 THEN
      mult[multiplicand(0),multiplier(i),toutpp(((i-1)*bits)+1),
        carryin] -> result(i), tcout(i*bits)
    ELSE IF j = bits-1 THEN
      mult[multiplicand(bits-1),multiplier(i),nextpp(i-1),
        tcout((i*bits)+j-1)] ->
        toutpp((i*bits)+j),
      nextpp(i)
    ELSE
      mult[multiplicand(j),multiplier(i),toutpp(((i-1)*bits)+(j+1)),
        tcout((i*bits)+j-1)] ->
        toutpp((i*bits)+j),tcout((i*bits)+j)
    ENDIF
  ENDFOR
ENDFOR

```

```

        ENDIF
    REPEAT
    REPEAT
    FOR j = 1:bits-1 CYCLE
        toutpp(((bits-1)*bits)+j) -> result(bits+(j-1))
    REPEAT
    nextpp(bits-1) -> result((2*bits)-1)

End

( ----- )

Part cas[onedivisor,onedividend,pr,cin] -> oneremainder,cdivout
    fulladder[mxor12[onedivisor,pr],onedividend,cin] ->
oneremainder,cdivout
End

( ----- )

Part varsquareroot (bits) [input(1:2*bits)] -> result(1:bits)

Integer count, complete, i, edge, index
Integer downmax, downcount, long

long = 2*bits

Signal oppsig(0:bits), down(0:bits)
Signal tcout(0:long*long-1)
Signal toutpp(0:long*long-1)

VDD -> oppsig(0)
downmax = 0
downcount = 0
complete = 0
index = 0
FOR i = 0:bits-1 CYCLE

    complete = complete + 2
    count = complete
    edge = 1

    WHILE ( count-1 >= index ) CYCLE
        IF count = complete THEN
            cas[oppsig(i),oppsig(0),input(complete),oppsig(i)] ->
toutpp((i*long)+complete-edge),tcout((i*long)+complete-edge)
        ELSE IF (count = complete-1) THEN
            cas[oppsig(i),minv01[oppsig(i)],input(complete-1),
            tcout((i*long)+complete-edge+1)] ->

toutpp((i*long)+complete-edge),tcout((i*long)+complete-edge)
        ELSE IF (count = complete-2) THEN

cas[oppsig(i),oppsig(i),toutpp(((i-1)*long)+complete-edge),
            tcout((i*long)+complete-edge+1)] ->

toutpp((i*long)+complete-edge),tcout((i*long)+complete-edge)
            downmax = downmax + 1
            downcount = downmax
            down(downmax) -> oppsig(i)
        ELSE
            cas[oppsig(i),down(downcount-1),

```



```

        toutpp(((i-1)*long)+complete-edge),
        tcout((i*long)+complete-edge+1)] ->
toutpp((i*long)+complete-edge), tcout((i*long)+complete-edge)
        downcount = downcount - 1
        ENDIF
        count = count - 1
        edge = edge + 1
        REPEAT
        tcout(i*long+index) -> result(i+1)
        tcout(i*long+index) -> oppsig(i+1)
        toutpp((i*long)+index) -> --
        index = index + 1
        REPEAT
        FOR i = 0:bits-1 CYCLE
        toutpp(((bits-1)*long)+i+index) -> --
        REPEAT
End
( ----- )

Part pass8[en,buss(1:bits),bussout(1:bits)] -> dum

        Integer i

        FOR i = 1:bits CYCLE
        mpas02[en,buss(i),bussout(i)] -> --
        REPEAT
        en -> dum
End
( ----- )

Part reg8[en,bus(1:bits)] -> busout(1:bits)

        Signal enb
        Integer i

        minv03[en] -> enb
        FOR i = 1:bits CYCLE
        mdff01[bus(i),enb,en] -> busout(i),--
        REPEAT
End
( ----- )

Part varadd(count) [a(1:count),b(1:count)] -> c(1:count), cout

        Signal cin(0:count), couttemp(1:count)
        Integer i
        GND -> cin(0)
        FOR i = 1:count CYCLE
        fulladder[a(i),b(i),cin(i-1)] -> c(i),couttemp(i)
        couttemp(i) -> cin(i)
        REPEAT
        couttemp(count) -> cout
End
( ----- )

Part sa[sb, bi,cin] -> sum, cout
        minv01[mnan02[cin,mxor12[sb,bi]]] -> cout

```

```

    mxor12[cin,mxor12[sb,bi]] -> sum
End

{ ----- }

Part tsas[sb,satemp(1:8)] -> saout(0:8)

    Integer i
    Signal coutmp(1:8), cin(0:8)

    sb -> cin(0)
    FOR i = 1:8 CYCLE
        sa[sb,satemp(i),cin(i-1)] -> saout(i-1), coutmp(i)
        coutmp(i) -> cin(i)
    REPEAT
        mxor12[sb,coutmp(8)] -> saout(8)
End

{ ----- }
{ ----- }

{ main program starts here }

{ ----- }
{ ----- }

varmult(bits) [m1(1:bits),m2(1:bits)] -> multbus(1:2*bits)
varsquareroot(bits) [add16(1:2*bits)] -> sqroot(1:bits)
varadd(16) [a16(1:16),multbus(1:16)] -> add16(1:16),--
varadd(8) [sqroot(1:8),a8(1:8)] -> addbus(1:8),--
reg8[minv01[mnor04[VDD,conbus(1),conbus(3),conbus(8)]],multbus(1:8)]
-> a16(1:8)

reg8[minv01[mnor04[VDD,conbus(1),conbus(3),conbus(8)]],multbus(9:16)]
-> a16(9:16)
reg8[minv01[mnor04[conbus(2:5)],bus2(1:8)]] -> m1(1:8)
reg8[minv01[mnor04[VDD,conbus(2:3),conbus(7)]],bus3(1:8)] -> m2(1:8)
reg8[consel,divbus(1:8)] -> a8(1:8)
tsas[sb,a16(1:8)] -> --,xy(1:8)
mdff01[sy,minv01[conbus(8)],conbus(8)] -> syl,--
mdff01[sx,minv01[conbus(8)],conbus(8)] -> sxl,--
mpas02[conbus(8),sb,syl] -> --
mpas02[conbus(1),sb,sxl] -> --
pass8[conload,outbus(1:8),divbus(1:8)] -> --

minv01[mnan02[minv01[minv01[minv01[minv01[conbus(5)]]]],minv01[conload]]]
-> en1
pass8[en1,divbus(1:8),addbus(1:8)] -> --
pass8[minv01[mnor04[conbus(1:4)],bus2(1:8),div1bus(1:8)] -> --

pass8[minv01[mnor04[VDD,conbus(1:2),conbus(6)]],bus3(1:8),div1bus(1:8)]
-> --
pass8[outsel1,m2(1:8),out1bus(1:8)] -> --
pass8[outsel2,xy(1:8),out1bus(1:8)] -> --

{ ----- }

{ IO PAD declaration is made in the main program.

{ ----- }

iopad1[minv01[mnor02[conbus(5),conload]],divbus(1),div(1)] ->

```

```

divlbus(1),divl(1)
  iopad2[minv01[mnor02[conbus(5),conload]],divbus(2),div(2)] ->
divlbus(2),divl(2)
  iopad3[minv01[mnor02[conbus(5),conload]],divbus(3),div(3)] ->
divlbus(3),divl(3)
  iopad4[minv01[mnor02[conbus(5),conload]],divbus(4),div(4)] ->
divlbus(4),divl(4)
  iopad5[minv01[mnor02[conbus(5),conload]],divbus(5),div(5)] ->
divlbus(5),divl(5)
  iopad6[minv01[mnor02[conbus(5),conload]],divbus(6),div(6)] ->
divlbus(6),divl(6)
  iopad7[minv01[mnor02[conbus(5),conload]],divbus(7),div(7)] ->
divlbus(7),divl(7)
  iopad8[minv01[mnor02[conbus(5),conload]],divbus(8),div(8)] ->
divlbus(8),divl(8)
  iopad9[minv01[conload],outlbus(1),out(1)] -> outbus(1),outl(1)
  iopad10[minv01[conload],outlbus(2),out(2)] -> outbus(2),outl(2)
  iopad11[minv01[conload],outlbus(3),out(3)] -> outbus(3),outl(3)
  iopad12[minv01[conload],outlbus(4),out(4)] -> outbus(4),outl(4)
  iopad13[minv01[conload],outlbus(5),out(5)] -> outbus(5),outl(5)
  iopad14[minv01[conload],outlbus(6),out(6)] -> outbus(6),outl(6)
  iopad15[minv01[conload],outlbus(7),out(7)] -> outbus(7),outl(7)
  iopad16[minv01[conload],outlbus(8),out(8)] -> outbus(8),outl(8)

```

Endoffile

END

0|8·0|8·88

FIN