National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Vous êtes l'abonné principal

Vous êtes l'abonné principal

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.

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

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.

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

Canada

PROCESSOR CORE DESIGN FOR INSTRUMENT CONTROL

by
MICHAËL A. E. GAZIER, B.Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Engineering
Department of Electronics

Carleton University
Ottawa, Canada
April 29, 1993

© copyright
1993, Michaël A. E. Gazier

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file / Votre référence

Our file / Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-89866-6

Canada

Masters Theses
PUBLISH ABSTRACT ONLY
AGREEMENT

Abstract No. _____

INDICATE YOUR APPROVAL OF THIS AGREEMENT BY SIGNING ITEM #11

PERSONAL DATA (Please type or print)

1. Full name GARTNER
(last) MICHAEL A E
(first) (middle)
2. Year of birth 1965
3. Country of citizenship CANADA
4. Present mailing address _____
18 DEERFIELD DR #1602
NE PLAW, ONT. K2G 4L1

Future mailing address _____
30 C BENLEA DR
NE PLAW, ONT

Effective date 15/02/93 Telephone number (613) 763-2604

DO NOT WRITE IN THIS SPACE

Vol/Issue _____

School Code _____

Advisor _____

Abstract Length _____

TP Date _____

MASTERS DEGREE DATA

5. Full name of university conferring degree
CARLETON UNIVERSITY
6. Abbreviation for degree awarded M.ENG. 7. Date degree awarded _____

TITLE/SUBJECT DATA

8. Attach a copy of your thesis title page and abstract to this Agreement Form.
9. Subject Category for your thesis. Enter 4-digit code from list on opposite side.

0544

ABSTRACT REPRINTS

10. Please send _____ reprints of my abstract. (Minimum order 100 copies)
First set of 100 \$ 30.00
Additional sets of 100 @ \$5.00 each set..... \$ _____
Subtotal \$ _____
Sales tax for state of ship-to \$ _____
10% shipping and handling \$ _____
Enclosed please find my check for..... TOTAL\$ _____

This is an agreement between the author and University Microfilms Inc., Ann Arbor, Michigan, providing for the publication of an abstract (of not more than 150 words) of the thesis in *Masters Abstracts International*, which is distributed on a subscription basis. If more than 150 words are contained in the abstract, it will be returned to the author for abridging. Mathematical formulas, diagrams, and other illustrative materials are not recommended for the printed abstract.

11. I agree to the provisions of this agreement.

Signature _____

Date 21.5.93

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

PROCESSOR CORE DESIGN FOR INSTRUMENT CONTROL

submitted by
Michaël A.E. Gazier, B.Eng.,

in partial fulfilment of the requirements
for the degree of Master of Engineering.

Chair, Department of Electronics

Thesis Supervisor

Carleton University

May 1993

ABSTRACT

Test instrumentation requires microprocessor based controllers as a bridge between analog electronics and large data collection and analysis systems. Other required functions may include control of mechanical components and processing of the acquired data. This thesis summarizes the results of design considerations for an integrated embedded instrumentation processor and focuses on processor core design. The application is a test system for semiconductor telecommunication lasers characterized in multi-laser bar form rather than in die form. Addressed processor core architecture design issues include RISC architecture, embedded processor requirements, pipelining, parallelism, caches, and DSP features. Other issues such as chip design methodology, testability, and performance measurement have also been addressed. A processor core design for instrument control has been developed. This design was simulated, the layout area estimated, and the design evaluated, providing a basis for synthesis of processor core designs for instrumentation control.

ACKNOWLEDGEMENTS

Completion of this thesis is a dream coming to fruition. The past years of part-time studies have been a challenging, rewarding, and important part of my life.

I would first like to express deep gratitude to my thesis supervisor Dr. Tad Kwasniewski for his wise and precise guidance. His patience and faith have enabled me to complete this work.

Over these years, my manager and friend Dr. Joe Finak has encouraged me in my thesis work, both on a personal level and by allowing me great flexibility within my profession.

I would like to thank my friends for understanding my commitment to my studies and not forgetting me, especially Michel. My colleagues and friends Ming and Jeff have given me courage whilst friends such as "Super-Dave" kept me lively with insane mountain-bike epics. My love Jackie has been especially supportive and understanding of my studies.

My parents Claude and Barbara have gifted me with the foundation to pursue this work and have encouraged me throughout my life to persevere and succeed. It is to them that I dedicate this work.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS.....	iv
TABLE OF CONTENTS.....	v
LIST OF ILLUSTRATIONS	viii
LIST OF TABLES	ix
LIST OF ABBREVIATIONS.....	x
CHAPTER 1. INTRODUCTION.....	1
1.1. THESIS MOTIVATION AND OBJECTIVES.....	1
1.2. THESIS OUTLINE.....	2
CHAPTER 2. DESIGN ISSUES FOR EMBEDDED CONTROL.....	3
2.1. DESCRIPTION OF APPLICATION.....	3
2.2. ARCHITECTURAL ISSUES	8
2.2.1. INTRODUCTION.....	8
2.2.2. SELECTED DESIGN ISSUES	8
2.2.2.1. RISC Technology	8
2.2.2.2. External Bus Interface.....	18
2.2.2.3. Interrupts and Exceptions.....	19
2.2.2.4. von Neumann and Harvard Architectures	21
2.2.2.5. Embedded processors.....	21
2.2.2.6. Pipelining.....	25
2.2.2.7. Parallelism.....	27
2.2.2.8. Caches	29
2.2.2.9. DSP issues	31
2.2.2.10. Other Design Issues.....	32
2.2.3. SUMMARY	34
2.3. CHIP DESIGN METHODOLOGY	34
2.3.1. INTRODUCTION	34
2.3.2. CASE STUDIES.....	35
2.3.2.1. Development of the TX1.....	35
2.3.2.2. Development of the MC68040.....	36
2.3.2.3. Development of the MIPS Processors	37
2.3.2.4. Development of the PA-RISC 1.1.....	37
2.3.3. SUMMARY	39

2.4. TESTABILITY ISSUES	39
2.4.1. INTRODUCTION	39
2.4.2. SCAN METHODOLOGIES	40
2.4.3. SELF TEST AND DIAGNOSTIC METHODOLOGIES	41
2.4.4. SUMMARY	43
2.5. MEASURING PERFORMANCE OF A PROCESSOR.....	45
2.5.1. INTRODUCTION	45
2.5.2. MEASURING PROCESSOR PERFORMANCE	46
2.5.2.1. Method Number One and Its Analysis	46
2.5.2.2. Method Number Two With an Example	47
2.5.2.3. Method Number Three With an Example	47
2.5.2.4. Performance and Chip Technology.....	49
2.5.3. OTHER FACTORS AFFECTING VALUE.....	49
2.5.4. SUMMARY	50
2.6. SUMMARY AND CONCLUSIONS	51
CHAPTER 3. DESIGN ARCHITECTURE.....	53
3.1. INTRODUCTION TO THE DESIGN.....	53
3.2. DESIGN DESCRIPTION.....	54
3.2.1. Ideal Design.....	54
3.2.2. Actual Design with Performance Evaluation.....	58
CHAPTER 4. DETAILED DESIGN AND VERIFICATION	65
4.1. COMPUTER DESIGN ENVIRONMENT.....	65
4.2. DETAILED DESIGN	66
4.2.1. System Modules.....	66
4.2.1.1. Opcodes.....	66
4.2.1.2. MicroProgram Controller.....	68
4.2.1.3. Branch Control Unit.....	76
4.2.1.4. Memory Facility	77
4.2.1.5. Arithmetic Logic Unit	88
4.2.1.6. Bus Interface Controller	94
4.2.1.7. EPROM and Clock units	99
4.3. SIMULATION AND LAYOUT OF DESIGN.....	100
4.3.1. Introduction.....	100
4.3.2. System Units	101
4.3.2.1. Branch Control Unit.....	101
4.3.2.2. Memory Facility Simulation.....	102

4.3.2.3. Arithmetic Logic Unit Simulation.....	102
4.3.2.4. Fault Simulation of ALU.....	103
4.3.2.5. EPROM and Clock Simulation	104
4.3.2.6. Bus Interface Controller	105
4.3.2.7. MicroProgram Controller.....	105
4.3.2.8. System Simulation.....	106
4.3.3. Layout Area Estimates and Evaluation.....	107
4.4. CONCLUSIONS ON THE DESIGN.....	109
CHAPTER 5. CONCLUSIONS AND FUTURE WORK.....	110
5.1. THESIS SUMMARY AND CONCLUSIONS	110
5.2. FUTURE WORK	112
REFERENCES.....	114
BIBLIOGRAPHY.....	116
APPENDICES	
Appendix A Design Files	
Appendix A1 -Design file types	135
Appendix A2 -Design File Names.....	136
Appendix B - Calculations for Chip Area Estimates	138
Appendix C - Original MC68000 application code.....	141
Appendix D - Sample plots	
Appendix D1 -Plot of ALU circuit diagram	157
Appendix D2 -Plot of ALUS simulation.....	158

LIST OF ILLUSTRATIONS

Figure 1,	BarTester System Overview	6
Figure 2,	BarTester System Architecture	9
Figure 3,	MicroEngine Architectures [6, chapter 7]	13
Figure 4,	Dual-Operation Paths in the i860 FPU [8]	16
Figure 5,	von Neumann and Harvard Architectures [15]	22
Figure 6,	A Generic Modular MicroController Chip, Adapted From [16]	24
Figure 7,	Example of Pipeline Usage and Associated Calculations	26
Figure 8,	Structure of the SRL and Its Use in an LSSD Network [28]	42
Figure 9,	“Eight Bit Bilbo Schematic”, Adapted From [28]	44
Figure 10,	Sample Kiviat Graph [29]	48
Figure 11,	BarTester System Architecture	55
Figure 12,	System Architecture Block Diagram	59
Figure 13,	System Architecture Overview	63
Figure 14,	System Architecture with Signals	74
Figure 15,	RIF Model Timing and Circuitry P1 of 2	82
Figure 16,	RIF Model Circuitry P2 of 2	83
Figure 17,	DPSRAM Read and Write Cycle Timing	86
Figure 18,	ALU Detail P1 of 2	91
Figure 19,	ALU Detail P2 of 2	92
Figure 20,	Timing of and Address Bus Latching in DVI, P1 of 2	96
Figure 21,	Circuitry of DVI, P2 of 2	97

LIST OF TABLES

Table 1,	Pinout of Processor	62
Table 2,	Opcode Functionality Breakdown.....	68
Table 3,	Vector Number Coding.....	69
Table 4A,	Listing of Processor Opcodes, Part 1 of 2	70
Table 4B,	Listing of Processor Opcodes, Part 2 of 2	71
Table 4C,	Description of Nomenclature Used in Table 4A and Table 4B.....	71
Table 5A,	Description of Signals To and From the MPC Unit, Part 1 of 2.....	75
Table 5B,	Description of Signals To and From the MPC Unit, Part 2 of 2.....	76
Table 6,	Description of Signals To and From the BRA Unit	76
Table 7A,	Hardware Implementation of the BRA Unit, Part 1 of 2.....	78
Table 7B,	Hardware Implementation of the BRA Unit, Part 2 of 2.....	79
Table 8,	On-chip DPSRAM Memory Map.....	80
Table 9,	Description of Signals To and From the RIM Unit.....	84
Table 10,	Description of Signals To and From the RIF Unit.....	85
Table 11,	DPSRAM Operation Modes.....	87
Table 12,	DPSRAM Timing Specifications for Port A	88
Table 13,	Description of Signals To and From the ALU Unit.....	93
Table 14,	Description of Signals To and From the AIF Unit.....	94
Table 15,	Description of Signals To and From the DVI Unit.	99
Table 16,	Description of Signals To and From the EPROM Unit.....	100
Table 17,	Description of Signals To and From the CKC Unit.....	100
Table 18,	Program for System Simulation	106
Table 19,	Manual Estimate of Chip Area	108
Table 20,	Manual Estimate of Number of Transistors in Design	108
Table 21,	Description of Design File Types.....	135

LIST OF ABBREVIATIONS

AC	-	Alternating Current
ADC	-	Analog to Digital Converter
AIF	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
ALLM	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
ALU	-	Arithmetic Logic Unit
ALU	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
ASIC	-	Application Specific Integration Circuit
BarTester	-	A test system built by the author with discrete components to characterize semiconductor lasers
BatMOS	-	An NT proprietary IC process to fabricate BiCMOS circuitry
BiCMOS	-	An IC process combining both Bipolar and CMOS circuitry
Bidco	-	Built in digital circuit observer
Bilbo	-	Built in logic block analyzer
BIST	-	Built In Self Test
Bit	-	The fundamental unit in digital systems, 1 or 0
BRA	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
Byte	-	A collection of 8 bits
CAD	-	Computer Aided Design
CISC	-	Complex Instruction Set Computer
CKC	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
CLASSIC	-	A proprietary NT software package for automated DPSRAM generation
CMOS	-	Complementary Metal Oxide Semiconductor
CMOS4	-	An NT proprietary IC process to fabricate CMOS circuitry
CMOS4S	-	An NT proprietary IC process to fabricate CMOS circuitry
CPI	-	Clocks Per Instruction

CPU	-	Central Processing Unit
DAC	-	Digital to Analog Converter
DMA	-	Direct Memory Access
DPSRAM	-	Dual Port SRAM
DSP	-	Digital Signal Processor
DVI	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
EPROM	-	Erasable Programmable Read Only Memory
EPROM	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
FML2	-	A proprietary NT software package for functional modeling
FPU	-	Floating Point Unit
FUNDES	-	A proprietary NT software package for functional design
FUNSIM	-	A proprietary NT software package for functional simulation
GPIB	-	General Purpose Interface Bus meeting standard IEEE-488
HOLIXZ	-	A proprietary NT software package for test vector description
IO	-	Input / Output
IC	-	Integrated Circuit
IR	-	Instruction Register
IU	-	Instruction Unit
KByte	-	KiloByte
KWord	-	KiloWord
LFSR	-	Linear Feedback Shift Register
LSB	-	Least Significant Bit
LSSD	-	Level Sensitive Scan Design
MAC	-	Multiply And Accumulate
MCU	-	MicroComputer Unit
MHz	-	MegaHerz
MIPS	-	Million Instructions Per Second
MISR	-	Multiple Input Signature Register
MMU	-	Memory Management Unit
MPC	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
MSB	-	Most Significant Bit
NT	-	Northern Telecom
O(f(x))	-	Order of function f(x)

PC	-	Program Counter
pF	-	picoFahrad
PLA	-	Programmable Logic Array
PRPG	-	Pseudo-Random Pattern Generator
RAM	-	Random Access Memory
RAM	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
RIF	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
RIM	-	Module in Experimental Work, see chapter "Detailed Design and Verification"
RISC	-	Reduced Instruction Set Computer
ROM	-	Read Only Memory
RTL	-	Resistor Transistor Logic
SA0	-	Stuck at zero fault
SA1	-	Stuck at one fault
SIM	-	System Integration Module
SIMD	-	Single Instruction Multiple Data
SP	-	Stack Pointer
SRAM	-	Static RAM
SRL	-	Shift Register Logic
SUDS	-	Standard Cell User-Designed Silicon (component library)
SYNFUL	-	A proprietary NT software package for sequential logic synthesis
TPU	-	Time Processor Unit
TTL	-	Transistor Transistor Logic
VLIW	-	Very Long Instruction Word
VLSI	-	Very Large Scale Integration
Word	-	Generally equated to two bytes, or 16 bits. May be used to define an arbitrarily long collection of bits although other terms such as Long-Word (32 bits) may be used for this purpose

CHAPTER 1. INTRODUCTION

1.1. THESIS MOTIVATION AND OBJECTIVES

Due to part volume increase and limited testing resources, a new test instrument was developed by the author for the optoelectronics device group at Northern Telecom, and implemented into production by mid 1989. The purpose of this instrument is to characterise semiconductor telecommunication lasers. Its peculiarity is that instead of the usual test method consisting of full characterization of one laser chip at a time with a low duty-cycle to allow cooling of the device during test, this system tests 10 lasers in parallel by interleaving the laser cooling time of each laser with test pulses to the other 9 lasers. This requires real time control of sequencers, ADC, DAC, power control, and communications hardware. Electronics apply a 5 microsecond 0.1% duty cycle pulse train to the laser and need a 500 nanosecond timing resolution to properly sequence the hardware. The resulting data is sent back to the main computer for analysis and storage before the next test cycle begins. Control signals are needed to operate mechanical systems such as relays and solenoids. Thus, "intensive" low-level control is needed to operate this system in its current configuration.

This thesis summarizes results of considerations for data and control path design in such an integrated embedded instrumentation processor. The design shares the discrete component hardware instrumentation application designed by the author at Northern Telecom. The integrated processor activity, however, is not related to the author's responsibilities at Northern Telecom. Design topics investigated include RISC architectures, interrupt handling, embedded processor requirements, pipelining,

parallelism, caches, and DSP features. Case studies on chip design methodologies are presented. Testability issues such as scan methodologies and self test and diagnostic are reviewed. Three methods for measurement of chip performance are presented. Performance attained by the designed integrated processor must be of the same order as the discrete component hardware described for the application. An open architecture allowing future upgrades and having a small on-chip static RAM for data storage are desirable. Although embedded processors are currently available on the market, this thesis presents a low-cost architecture designed to the gate level, as well as an approach for integration of embedded instrumentation control features. The thesis does not dwell on the design of the sequencer and other custom units, as the open architecture designed easily accommodates such components. Although an understanding of testability issues is required and general concepts applied, incorporation of full testability circuitry in the chip design is not within the scope of the thesis. Layout and fabrication of the design as a silicon device are discussed but not implemented as these again are outside the scope of the thesis.

1.2. THESIS OUTLINE

Chapter 1 provides an introduction including thesis motivation and objectives and an outline to the thesis. Design issues for embedded control are discussed in chapter 2, and specifications for the embedded instrument processor are further detailed in section 2.1. In order to successfully implement such a processor, a literature review on embedded processors is presented in sections 2.2 to 2.5 as follows. Section 2.2 introduces selected topics for embedded instrumentation processor design including a RISC architectural highlights, especially with respect to traditional CISC architectures. Bus interfaces, interrupt requirements, parallelism, caches, and topics specific to embedded processors are then covered. Digital signal processors and other various design issues are also examined. Section 2.3 examines chip design methodologies by reviewing case studies for four processors. Chip testability is discussed in section 2.4 with an overview of design for testability, LSSD scan-chains, and Built-In Self Test BIST. Section 2.5 examines the issue of measurement of the value of a chip, wherein chip performance measurement is discussed as well as other issues affecting chip value such as cost, suitability to an application, and functionality. Chapters 3 and 4 describe design and verification of a processor meeting design specifications as per section 2.1. In chapter 3, an ideal design is described as derived from the literature search and an actual design is introduced. Chapter 4 describes the detailed design, with its simulation and evaluation. Finally, layout area estimations for this processor are presented. Chapter 5 concludes the thesis and offers suggestions for future research in this area.

CHAPTER 2. DESIGN ISSUES FOR EMBEDDED CONTROL

2.1. DESCRIPTION OF APPLICATION

Due to part volume increase and limited testing resources, a new test instrument was developed by the author for the optoelectronics device group at Northern Telecom, and implemented into production by mid 1989. The purpose of this instrument is to characterise the light output power versus current of semiconductor $1.3\mu\text{m}$ and $1.55\mu\text{m}$ III-V InGaAsP telecommunication lasers. The peculiarity of this instrument is that instead of the usual test method consisting of full characterization of one laser chip at a time with a 0.1% duty-cycle to allow cooling of the device during testing, this system tests 10 lasers in parallel by interleaving the laser cooling time of each laser with test pulses to the other 9 lasers.

This test is possible as a result of several factors. First, the devices are presented in bar form rather than in chip form to the tester. Previously, each wafer of semiconductor lasers was scribed into bars of approximately 40 lasers each. Each bar was then cleaved into 40 die, or individual devices, which were then tested. A die measures about 10 by 15 thousandths of an inch and is difficult to manipulate. To test a die, it must be loaded into a test jig, the lasing facet aligned to the jig orientation, and the contact pad on the top surface aligned to two probes. One probe is for current injection and the second for voltage sensing. Then, a pulsed current ramp is applied while the device's electrical and optical responses are captured. Generally, it is desirable to digitize this data and transfer it to a computer for analysis of various parameters such as resistance, lasing threshold current,

and whether the device exhibits non-linearities referred to as kinks. Once analyzed, the pass devices must be sorted from the fail devices.

The so-called **BarTester** system simplifies several steps in this process. Manual labour is reduced by testing the lasers in bar form rather than in die form, as bars are much bigger and thus easier to handle than the individual die. Bars are placed four at a time on a test jig allowing photodetectors to access both front facet and rear facet light emissions of the lasers. The jig contains a groove with a vacuum such that the bars are automatically centered and held in place. This jig is placed on a motorized stage which can move between the four areas of bar loading, picture acquisition, probing, and inking. After the bars have been loaded into the jig, the stage moves the approximate edge of each bar under a picture acquisition area where a camera takes a snapshot and sends the image to the computer for determination of the probing area of the first laser in the bar. This process is repeated for each bar on the jig. With this information, the computer can instruct the stage to align the bars perfectly under the probing area. The probes are a set of 10 pairs of probes spaced to correspond to the lasers' contact areas' pitch. Of each set, one probe is used to inject the current into the laser and one to sense the resulting voltage. Using two probes avoids the voltage drop due to cable resistance and test current since the voltage sensing requires negligible current. This whole alignment process removes contact area alignment and individual device manipulation by the test system operator. In order to test them properly, the devices on the bar must be isolated from each other, either by the device structure or by a specific process step such as a trench isolation. Testing of semiconductor laser devices requires a 0.1% duty cycle pulse train so as not to overheat the devices. Thus for a 5 microsecond pulse applied to the laser, the test system must wait 5 milliseconds before applying the next pulse. The **BarTester** probes 10 lasers simultaneously and thus can pulse the other 9 lasers during the cooling time required for the first laser. In fact, enough time remains after pulsing all 10 lasers before the next current pulse that the **BarTester** uses this time to send the acquired data to the main data storage computer. Thus, the normal system approach of testing one laser at a time is accelerated 10 times simply by interleaving device tests. Other accelerations ensue by uploading the data to the data storage computer simultaneously to test rather than after characterization and by uploading data packets to the data storage computer. This latter feature has the effect of engaging a DMA transfer mode on the main computer, thus accelerating data transfer even more. Finally, the system test has been sped up by accelerating the pulse timing from 10 microseconds in previous systems to 5 microseconds.

All these factors bring tremendous speed improvements to device testing. A last improvement ensues from the use of an inker. Although commonly used on silicon device testers, die inking is not used in laser device testers due to the small size of these devices. Since the **BarTester** tests devices in bars of approximately 40 devices, inking has been incorporated as final step before the devices are unloaded. As devices are cleaved into individual die, they are easily sorted into pass and fail devices. Other advantages of bar-testing are not presented as they are proprietary to specific Northern Telecom processes.

It can be seen that this type of testing requires real time control of sequencers, ADCs, DACs, and analog electronics. Power control is required to switch vacuum and nitrogen solenoids and the inker for the system's mechanics. Communications hardware between the system and the data storage computer are also needed. It may be beneficial to have some local computation power to perform data manipulation such as data smoothing. It can be seen that "intensive" low-level control is needed to operate this system as defined.

Figure 1 outlines the system architecture implemented. The laser bar to be characterized is shown as **Bar of 40 Lasers**. The **Data Analysis And DataBase Computer** controls the entire system. System mechanics are labelled **Physical System (Stages, Probes,...)**. This includes the main stage with a bar holding jig and a 20 pin probe card, relays for the inker, solenoids for vacuum and nitrogen control. The test sub-system is composed of a digital system, **Processor Boards**, an interface between the digital and analog systems, **Analog MicroControl Board**, and an analog section labelled **Analog, DAC, ADC Boards**. This latter applies the current pulse to the laser under test, switches the current to and the voltage back from the laser, drives the photodetectors, and includes the analog-to-digital converters. Figure 2 details these systems in more detail. The **Main Processor Boards** include two interfaces to the **Database And Data Analysis Computer**, namely a standard GPIB interface and a standard GPIO interface. The processor used is a Motorola 68000 operating at 10 MHz with 8 KWords of static RAM and 8 KWords of EPROM and minor glue-logic. Circuitry to allow the processor to control I/O at high voltages and currents is interfaced with a parallel interface/timer and denoted by **Parallel Power Control**. The timer is used both for proper timing of the inker control and to ensure the 0.1% duty cycle for the pulse train. The picture acquisition camera uses outputs an encoded TTL level signal and is decoded by the **Custom Camera Interface board**. The **Auxilliary Analog Micro-Control Digital Board** includes some logic for interfacing the main microprocessor to the ADCs, DACs, and multiplexers in the analog section as well as a microsequencer for sequencing of test operations.

Figure 1, BarTester System Overview

The **microsequencer** is a state machine which generates precise timing signals for generating the 5 microsecond current pulse to the laser under test. The pulse sequence is to halt the system processor by withholding the data transfer acknowledge **DTACK** signal, switching the pulse to the proper laser away from the dummy load, waiting for the response to settle, triggering the sample-and-hold circuitry, triggering the ADCs, switching the pulse back to the dummy load for transition minimization, and releasing the **DTACK** signal. The main processor is responsible for providing the 0.1% duty cycle timing for the pulse train and in fact uses its timer for this purpose. The **Analog Boards** are composed of six main sections as follows. A **Current Source** is controlled by a 16 bit number from the main processor. A sixteen channel **Output Current Multiplexer** switches this current between a dummy-load, the ten probes to the device under test, or to various calibration loads. The **Input Voltage Multiplexer** switches the selected channel's voltage, including that from the ten laser voltage sensing probes, back to the measurement electronics. The multiplexed channel is preprogrammed into the **Analog Bus Address Decoder** and controlled by the **Microsequencer** under main processor control. The light emitted from the laser under test is collected by a **Front and Back Facet Detector**. These are driven by a **Front and Back Facet Detector Bias Circuitry** with an output voltage proportional to the captured optical power. This voltage is connected to the measurement electronics which are composed by a **Level Conditioner/Filter/Input Amplifier** stage followed by a **4-Channel 16-bit Digitizer**. The four channels are the current output to the device under test, the voltage sensed, and the front and back facet light detected. All devices such as timer, DACs, ADCs are memory mapped to the main processor. This means that the processor accesses the devices as if they were all a memory chip, with each device's control registers "mapped" to a particular memory location.

It has thus been proposed that the **Main Processor Board** without the EPROM and SRAM memories and the **Auxiliary Analog Micro-Control Digital Board** could be implemented as an integrated instrument controller chip. In addition, it would be desirable to have a small on-chip SRAM for data storage. Performance must be of the same order as the present described system, including the real-time sequencer. An open system allowing for future upgrades would also be desirable. This would lower system cost and complexity of the **BarTester** by integrating as much of the digital circuitry as possible. This thesis focuses on the design of a CPU core for a low-cost embedded instrumentation processor such as this one, but does not dwell on design of the sequencer and other custom units as the architecture is easily upgradable to accommodate such components.

2.2. ARCHITECTURAL ISSUES

2.2.1. INTRODUCTION

Design of a processor requires knowledge of processor architectural concepts, of semiconductor technology, and of software. In addition, design of an embedded instrumentation processor such as the one discussed in this paper requires knowledge about the requirements and limitations of embedded applications. Some important concepts are highlighted in the following chapter and are related to embedded instrumentation processor design. Topics reviewed from the appropriate literature include a review of RISC architectures since RISC is increasingly important in chip design and provides relevant advantages over traditional CISC architectures. A review of bus interfaces, interrupt requirements, and von Neumann versus Harvard architectures is then provided. Some issues relating specifically to embedded processors are then examined. Pipelining, parallelism and cache issues are discussed as these have migrated down to embedded instrumentation processors from earlier high-performance systems. Finally, digital signal processors DSP and other various design issues are examined. These, along with the issues of chip design methodologies, testability, and processor value as examined in the following chapters provide the basis for the design of an embedded instrumentation processor.

2.2.2. SELECTED DESIGN ISSUES

2.2.2.1. RISC Technology

The two main characteristics of RISC involve the use of a minimal instruction set and the execution of instructions in a single clock cycle. These allow RISC processors to efficiently emulate CISC processors by executing many simple operations in series. The RISC features shown below lead to improved performance and are discussed in the following pages.

CISC processors attempt to squeeze the most performance out of any given architecture by using the fact that an internal microcode executes much faster than external instructions. Thus the design goal for CISC is to build high semantic content instructions into the processor to reduce the number of external instructions the processor must fetch. In addition, a microcode allows the inclusion of many addressing modes and several control points in the processor's internal hardware. CISC processors allow any instruction, not only load/store, to access memory when useful. Due to circuit complexities in CISC processors, much more time is needed to design and develop this type of processor than a RISC processor. RISC processors, however require more design care to ensure higher clock rate operation.

Figure 2, BarTester System Architecture

Most RISC architectures fall into one of two competing strategies. One is based on work done at the University of California at Berkeley, the other on research at Stanford University [1]. The basic architectural feature of the Berkeley approach is the large use of windowed registers, a leading example being the SPARC architecture is used by SUN. The other major RISC architecture is the Stanford approach which uses fewer registers and relies on the compiler to optimize execution by placing instructions efficiently into a pipeline. An example of the latter is found in the MIPS architecture R3000 processor which follows the Harvard model, with separate data and address internal buses, and a physical cache rather than a virtual cache, which simplifies multiprocessor system implementation [1].

The following are considered the main features of RISC technology and will be presented in the following sections:

- Single cycle operation for most instructions
- Load/store interface to memory and register based execution
- Large register set or register windows
- Hardcoded Control Pod
- Relatively few instructions and addressing modes
- Fixed instruction format and length
- Larger compile time effort
- Minimum pipelining but parallel internal execution
- More concurrency visible to software (eg: branch structures)

Single-cycle operation for most instructions

In order to operate in a single cycle, the instruction must either be simple or be backed by special hardware. Instruction throughput is measured in millions of instructions per second MIPS. A high MIPS number does not solely determine how powerful the processor is, because MIPS does not measure the work accomplished but only the number of actions accomplished. It is to be noted that by loading several instructions worth of data per cycle a throughput of more than one instruction per cycle may be possible, effectively yielding fractional cycle times.

Load/store interface to memory and register-based execution

Only load and store instructions access external memory. This allows all other operations to operate in a single cycle as they will involve register to register transfers only, not transacting with external systems such as memory which have an indeterminate number of cycles to access.

Because registers are intensively used in such a load/store architecture, the high cost of fast memory such as SRAMs may be justified.

Large register set or register windows

The number of user accessible registers is a key factor in the performance of RISC processors, given the relative performance penalties associated with accessing variables in cache or in memory. This and the fact that modern compilers have increasingly sophisticated register allocation schemes drives a large set of registers to be built in to today's architectures.

Studies performed to determine how many registers are necessary in processors using stack based parameter passing were explored in developing the i376 [2]. It was determined that 90 percent of the time eight registers were sufficient for assembly language programs, and 98 percent of the time they were sufficient for optimizing compilers. Expanding to 16 registers increases performance by 5 to 15 percent. As well, it was found that allocating registers across procedures was useful, while parameter passing through registers required at least 32 registers to be effective.

Studies on register-windowing schemes as opposed to sophisticated register allocation techniques have shown that the relative performance of both these techniques is essentially equal but that register-windowing offers better performance in certain cases [3]. However not all cases in this study used the advantages of the register-windowing scheme due to the newer more sophisticated approaches to register allocation that take advantage of programs characteristics such as the high percentage of time spent in leaf procedures. Also, compiler register allocation may slow down the compiling speed to unacceptably low levels [4].

HardCoded Control Pod

Refer to Figure 3 for a generalized block diagram of the two possible architectures, namely hardcoded and microcoded. Microcoded architectures have variable-length instruction size and execution time. Thus, even though microcode control is more flexible and uses less silicon area than a hardcoded control, microcode can not easily guarantee single cycle operation and so is not used in order to maximize performance.

Relatively few instructions and addressing modes

Relatively few instructions and addressing modes facilitate implementation of both single-cycle operation and hardcoded control with a relatively small investment in design time and silicon real estate. Also, more easily decoded instructions and simpler addressing modes can yield faster execution times. Sun's C compiler uses only 30% of available MC68020 instructions, and studies have shown that about 80% of computations for a typical program require only about 20% of a processor's instruction set [4].

The i860, MC88000, and SPARC architectures have two basic addressing modes which are (base + offset) and (base + index). Since register 0 contains a constant 0 value, five addressing modes can be synthesized. With x and y denoting a register number, the addressing modes are register R_x , register indirect (R_x), register indirect with index (R_x, R_y), register indirect with immediate offset $offset(R_x)$, and immediate offset, signed and unsigned. The arithmetic operations in the i860 always set the condition codes, unlike the MC88000 and the SPARC architectures. This renders implementation of complex pipelining schemes supporting out-of-order execution and multiple-instruction executions difficult. The MC88000 writes status information resulting from a compare instruction in a general register specified in the compare instruction; correspondingly conditional branches test the specified general register instead of the traditional processor status register. This allows the MC88000 to lend itself much more easily to the complex pipelining schemes.

Sample HardCoded Block Architecture

Sample MicroCoded Block Architecture

Figure 3, MicroEngine Architectures [6, chapter 7]

Fixed instruction format and length

By issuing instructions of different sizes it is possible to reduce the amount of memory necessary to store a program and thus to reduce the necessary bus bandwidth as well as cache size. Programs will also load faster into the processor [5]. On the other hand, fixed instruction formats and length simplify design of the control circuitry and also leads to faster execution and are thus used in pure RISC designs.

Larger compile-time effort

Much of the static run-time complexity can and should be handled prior to run time by an optimizing compiler. RISC strategies lead to the offloading of the more complex or infrequently used instructions onto the compiler. The instructions and addressing modes left are those frequently used by the compilers' code generators, those most advantageous to a language, those that are much more efficient if implemented in hardware such as cache units or barrel shifters, and those with special functionality such as communication hardware. This approach also enhances performance by moving decoding from run time to compile time since no further calculations will be required each time the programs runs. In addition, optimizing compilers can organize pipelined instruction sequences and arrange register-to-register operations to reuse computational results.

The following optimizations are quoted for Hewlett-Packard's PA-RISC processors [7].

- global data flow and alias analysis: knowing which data items are accessed by code and which data items may overlap; is the foundation for many phases that follow
- constant propagation: folding and substitution of constant computations
- loop invariant code motion: computations within a loop that yield the same result for every iteration
- strength reduction: replacing multiplication operands inside a loop with iterative addition operations
- redundant load elimination: elimination of loads when the current value is already contained in a register
- register promotion: promotion of a data item held in memory to one being held in a register

- common sub expression elimination: removal of redundant computations and the reuse of the one result
- peephole optimizations: use of a dictionary of equivalent instruction patterns to simplify instruction sequences
- dead code elimination: removal of code that will not execute
- branch optimizations: transformation of branch instruction sequences into more efficient instruction sequences
- branch delay slot scheduling: reordering instructions to perform computations in parallel with a branch
- graph colouring register allocation: use of a technique called graph colouring to optimize the use of machine registers
- instruction scheduling: reordering instructions within a basic block to minimize pipeline interlocks
- live variable analysis: removing instructions that compute values that are not needed

Minimal pipelining but parallel internal execution

Pipelining in a CISC allows more efficient use of the available bus bandwidth and allows performance equivalent to a RISC architecture. However, since the RISC processor has a turnaround time of one cycle per instruction execution, there is no point in implementing it in RISC processors. The savings in chip area and design time due to RISC concepts as well as the instruction set's structure allow for the processing of several instructions at the same time but require a complex compiler. Features such as pipelining and high-performance memories have been initially developed in supercomputer designs such those of Seymour Cray [1] [4] and adapted to RISC architectures by integrating them into a processor with a simple instruction set and no microcode.

It can be seen that with proper control, an architecture such as that shown in Figure 4 supports not only "medium grained" parallel operation of several units such as the FPU and the Integer Unit but also "fine grained" parallel operation within a unit. Using multiple functional units in RISC designs allows execution of several instructions per cycle. Since these must be in the instruction pipeline, branches may have effect on many instructions at a time.

Figure 4, Dual-Operation Paths in the i860 FPU [8]

Register scoreboarding allows multiple instructions to execute concurrently with a guarantee that stale data will never be used. Thus the processor may continue code execution unless a subsequent operation attempts to use data that is not available. An optimizing compiler may organize the code so that it is always executing.

More concurrency visible to the software

The design of the IBM RS/6000 increases use of the compiler and operating system in managing the hardware, thus not requiring a significant amount of hardware to provide maximum parallelism among functional units. This is accomplished through elaborate register dependency checks, branch prediction, branch history tables, store/load interlocks, and so forth [9]. Since the compiler software manages the hardware, this exposes parallelism in the processor to the compiler and operating system, requiring explicit management of the various units and their interactions in order to fully exploit the hardware, and even requiring software control to properly operate the processor.

Having the CPU pipeline stages more directly controlled by software as in the i860 has advantages as shown in this example of explicit pipelining. The software issues a new FP instruction to the first stage of the pipeline and gets back the result of the last stage of the pipeline every clock cycle, at which time the destination register is specified. This avoids tying up registers for results so that they may be used as part of the pipeline. Implicit pipelining, using scoreboarding, would cause the registers to become the bottleneck in the floating-point unit [8].

"Delayed branches" occur when the instruction sequentially following the branch executes independently of whether the branch is taken or not. This feature increases performance of the pipeline by reducing the flushing effect of branches on the pipeline. Studies have shown this technique is successful in eliminating the branch penalty in 60 to 70 percent of cases [?]. Another method of reducing branching penalties is to incorporate branch prediction in the instruction set as in the AT&T Crisp architecture, where the software uses two different branch instructions as to whether the loop is expected to occur or not. While studies have shown that software branch prediction may be more cost effective to implement, hardware schemes are not very expensive and do provide very good branch prediction [3]. Trace data from studies of actual code show that 75 percent of

branch instructions are used. Thus, the branch control in the MC68040 was optimized for cases when branching occurs [10].

When programming in a high-level language there is a 10 to 15% increase in memory requirements, but current memory prices make this a less significant factor. For example, when Ford compared a CISC with larger die size, slower throughput, and smaller memory size versus a RISC with smaller die, fast throughput, and a 25 to 100% larger memory size due to inherent RISC characteristics and programming a high-level language, the RISC approach was found to be much more cost effective [11][12].

2.2.2.2. External Bus Interface

Several types of external bus interfacing techniques are possible. In an asynchronous bus, bus events such as state changes or signal assertions may occur at any time. This is flexible, but requires handshaking to confirm data has been latched in. Connecting an asynchronous bus to the microprocessor's synchronous bus may cause metastability problems, but this may be solved by doubly latching the asynchronous input signals [13]. Thus the signal, once sampled, should have its sampled value latched later to resolve a potential metastability condition. This means more time must be dedicated to guaranteeing the state of the signal with an ensuing loss of bandwidth. In the synchronous bus, all bus events occur at specific times that are invariant with respect to clock signals. This is less flexible, but frees the designer from having to use handshake methods. The pseudosynchronous bus represents a combination between the last two options. For example, in the WE32100 [13], the I/O bus protocol is synchronous, whereas its method of indicating bus events is asynchronous, using strobe signals.

One of the major difficulties of having a non-multiplexed external bus system is that it requires a much greater chip pinout. In a non-multiplexed chip, the number of pins for data and address will be practically double that of the multiplexed version; if less than the internal bus's width is used then system performance degrades. This is a serious consideration both from cost and technology points of view, since the number of pins in a VLSI design partly determines in great part the integrated circuit's cost. One major drawback to multiplexing a bus between data and address lines is that most RAM and peripheral chips require both address and data to be present together, requiring off-chip glue-logic to have a complete system.

Parallel and very wide buses can be used on-chip without regard to pinout limitations, decoupling CPU cycle time and bandwidth on on-chip processing from off-chip considerations. Thus the processor may run at a very fast cycle time relative to the frequency of board signals and may have inexpensive packaging [14].

It is important to minimize the number of cycles the microprocessor has to wait for the data to be presented for a read cycle or accepted for a write cycle, the "wait states". An important parameter is the "zero wait-state memory access time", the maximum time that a peripheral can take to present data back to the microprocessor without incurring wait-states. A good read transaction protocol should maximize this time in order to ease the system designer's timing budget as well as to accommodate as many commercially available memories and peripherals as possible and thus the ratio of memory access time to total transaction time should approach one [13]. In fact, recent microprocessor based machines have similar peak CPU performance compared to mainframes but are less performant due to smaller and less performant memory and I/O subsystems [14].

Availability of a so-called background debugging mode in which the processor's registers may be read from or loaded with external data, and in which a single stepping mode and a breakpointing mode are provided, are very useful for system debugging. A 3-pin serial interface such as in the new family of MC683xx Motorola processors helps minimize pinout count, facilitates connection to the host debugging system and could possibly be integrated to LSSD scanning of the processor (see section 2.4.2).

2.2.2.3. Interrupts and Exceptions

There are two common ways for a processor to respond to an interrupt. Upon receiving the interrupt, the processor may initiate an interrupt vector number fetch cycle. This means that a read cycle is initiated after the interrupt is received with the "interrupt-vector" read from the bus. This is then used as a pointer to a "jump table" which in turn points to a segment of code to respond to the particular interrupt. The processor may also have priority encoded interrupt lines as separate pinouts and thus execution flow jumps directly to the appropriate interrupt vector fetched from the corresponding interrupt table pre-built into memory.

Interrupts are usually implemented to allow termination of the presently executing instruction. This makes saving the internal pre-interrupt state of the processor easier, but is still a complicated procedure if there is a pipeline in the processor. There are several possible software interrupts such as address trap break, execution step break, and flow trace. The execution step break function generates a self-debug exception after every execution of a specified number of instructions, allowing single step operation. The flow trace function is used to record the two most recent points at which the instruction execution sequence was interrupted. The i860 has a data breakpoint register that breaks upon matching the address to the register's contents.

It is critical in real-time embedded applications to respond both quickly and predictably to interrupts. In CISC processors this is handled in microcode and is slow, whereas in RISC processors the bulk of interrupt processing is left to the operating system or interrupt handler and thus can have very rapid response. For example if an MC68040 receives an interrupt it finishes the current instruction, conceivably a long one such as a multiply, initiates an interrupt acknowledge cycle to obtain an interrupt vector, finds the address pointed to in a table, makes an internal copy of its program counter and processor status, switches from user to supervisory mode, stores a four word stack frame to main memory and any registers used by the handler. If the handler is written in C the entire register set is saved. Then the MC68040 fetches the program counter from the exception table, and executes the handler. To complete the interrupt the opposite sequence is performed. If a higher level interrupt is received during interrupt handling the sequence will be repeated. It can be seen that calculating the processor's interrupt latency is difficult. In the RISC processor, the finer instruction granularity and shorter average execution time will enable it to respond more quickly. This granularity also makes the processor less likely to receive an interrupt during an instruction, minimizing instruction restarts, improving performance and simplifying memory system design. The RISC chips pay a higher penalty for pipeline flushes and have lower context switching speeds due to their large amount of registers needing to be saved prior to servicing an interrupt. The AMD29000 requires at least 192 clock cycles to save its 192 registers[12].

Register windows used by RISC devices such as the SPARC and i960CA help speed context switching by leaving the register contents intact and giving the process a new set of registers upon context switch. The i960CA provides 5 to 15 windows of 16 registers each on-chip. This will not work in interrupt-intensive processes such as telecommunication

switching where the maximum window number may be exceeded, requiring the processor to write register windows to main memory to make space.

2.2.2.4. von Neumann and Harvard Architectures

The so-called von Neumann bottleneck occurs when the microprocessor processes information faster than the memory system can supply the data. One way of relieving this is to implement a cache subsystem unit. The other approach is to use an alternate architecture, the most common being the Harvard architecture [17][15]. It is composed of two separate paths for data and instructions to the processor, each of which may have its own MMU and cache, see Figure 5. This Harvard architecture may be external to the processor such as in the MC88100 processor, or internal, such as in the MC68030 which has a single external bus but an internal Harvard type structure.

2.2.2.5. Embedded processors

Embedded controllers are a class of processors used on-board such self standing "intelligent" machines such as laser printers, facsimiles, numerical control machine tools, and industrial robots. A memory management unit (MMU) is not required in such applications and cache controllers or floating point units FPUs may be excluded for simplicity [5][12]. Such cuts reduce power consumption, facilitate use of low-cost plastic packaging, and make space available for other on-chip applications such as cache and high-speed DMA. It is estimated that over a four year period starting in 1989 the number of RISC chips sold to the embedded market will increase almost 20 times [1]. Thus the 200,000 embedded processors which constituted about 50% of the RISC chip market in 1989 were expected to make up about 75% of the market by 1992. Until recently, CISC processors such as the MC68000 family were used in embedded instrumentation processor applications but embedded instrumentation processors such as the i80960KA, i80860KB, AM29000, NS32CG16, and the NS32GX32 have been recently introduced to the market. They have clock speeds between 16 and 30 MHz, and run from 7 MIPS to over 20 MIPS.

Figure 5, von Neumann and Harvard Architectures [15]

High-performance computational capabilities, a large address space, and the ability to process large amounts of complex and high speed I/O are necessary for real time control, such as in embedded processors. Increasing the performance of a system requires more than an increase in CPU performance. A CPU performance increase may help in servicing high-speed I/O and communication channels but is not sufficient because of the required real-time response time to multiple interrupts. Since binary compatibility is not critical for embedded processors, the i376 embedded processor is not backward compatible with the 8086 or the 80286 processors [2]. This is because designers not only change the CPU in embedded applications but usually implement major hardware changes in an upgrade, requiring the software to be rewritten anyway.

The System Integration Module SIM in Figure 6 contains the interface to the external bus, including interrupt and bus error handling, address decoding pins, and control signals. The peripheral modules shown may contain intelligent controllers such as high speed serial controllers. The CPU represented contains a separate instruction execution unit with instruction pipeline and bus execution unit; the former executes instructions and effective address calculations while the latter increments the program counter. One way of accelerating instruction execution is to divide the instruction unit into separate address and data execution units, each with arithmetic capabilities. But an analysis of MC68000 family designs determined that the parallel execution of these separate units does not occur frequently when controlled by one microengine [16]; thus it was decided that a single instruction unit would be acceptable. On the other hand, a wide execution unit provides fast instruction execution, simplifies control, reduces signal routing, and simplifies design [16].

The Time Processor Unit (TPU) peripheral unit was developed to fill timing needs in the MC68332 CPU. CPUs cannot usually directly control timing applications such as input capture, output compare, pulse accumulation, and pulse width modulation without the help of additional hardware such as timer peripherals. The TPU is a microcontroller by itself and can thus perform timing tasks without CPU intervention, with an instruction set specifically tailored for timing tasks. The TPU is RISC-like in the sense that it has no expanded instructions and all instructions execute in one instruction cycle consisting of two system clock cycles. The device is service driven rather than interrupt driven. The TPU is composed of a host interface, a scheduler for deciding the order to service the signals, a microengine to sequence the instructions to service the signals, two timer count registers, and finally the timer channels themselves.

Figure 6, A Generic Modular MicroController Chip, Adapted From [16]

In embedded applications, designers must consider memory cost since many 32 bit embedded applications have up to 1 MByte of code and up to 2 MBytes of DRAM. Thus an increase in code size can significantly affect the overall cost of the system. Berkeley's RISC II requires 25 percent more space than a VAX, Sun's SPARC chip needs 50 percent more space than the MC68020, and a RISC subset of the VAX requires 250 percent more space than full VAX implementation. However, as the cost of memory drops the fact that RISC code takes more memory becomes less meaningful.

2.2.2.6. Pipelining

Pipelining has a time synchronism similar to Single Instruction Multiple Data SIMD parallel processing [17]. Pipelining is a system based on job subdivision with a precedence constraint. Partition of the workload is important in order to efficiently subdivide work into usable and efficient elements. "Synchro-Parallelism" denotes a symmetric partition in which the components are synchronized in time. This is a generalized form of SIMD parallel processing in which there is no explicit "instruction stream" [17]. There are precedence requirements within the pipeline in which there may be interdependencies between different tasks to be executed, and thus some tasks may have to wait for execution. Assembly line type timing may be used to keep the pipeline full while keeping the precedence requirements satisfied. There may also be "overlap" in which there is overlap between the processor, instruction preparation processing and execution, and the I/O unit. If the I/O unit is asynchronous, the processor may request an I/O transaction and then continue processing in parallel with the I/O unit executing it. This may not be possible in a von Neumann design where the I/O unit and processor may have to share common resources such as memory and data paths.

"Tight-coupling" occurs when all units in the pipeline are in synchro-parallelism, with work completion in one unit implying completion in all units. The efficiency of such a system for a particular job is calculated by determining the percentage of the maximum potential work done [17]. Refer to Figure 7 for further details. It can be seen:

$$\begin{aligned} \text{Efficiency} &= (\text{job area})/(\text{area swept by pipeline}) \\ \text{and: Performance} &= (\# \text{ stages in pipeline}) * (\text{Efficiency}) \end{aligned}$$

Figure 7, Example of Pipeline Usage and Associated Calculations

The possibility of data interactions between stages of a pipeline increases with pipeline length. This interaction increases the probability of idle stages and must be avoided for efficient usage of the pipeline. Interleaved instruction streams may maintain efficiency by providing alternative instructions to execute during pipeline stalls.

2.2.2.7. Parallelism

There are three levels of parallelism referred to as coarse, medium, and fine-grained. [18]. Fine-grained parallelism occurs at the instruction level when execution units on the CPU execute more than one instruction simultaneously. Medium-grained parallelism occurs at the loop level wherein a number of iterations of a particular loop can be mapped to a number of processors in the system, and this independently of the number of processors in the system. This is a breakdown of parallelism within a process. Coarse-grained parallelism occurs when the processes are mapped out to different processors.

Very Long Instruction Word VLIW processors rely heavily on sophisticated software to generate the code required to use machine resources effectively. The two companies which tried this approach failed but it has been suggested this was for non-technical reasons [19]. In the VLIW processor, the compiler takes responsibility for finding operations that can be issued together and for creating a single instruction containing these instructions. Because the compiler does this statically, the hardware does not need to check any dependencies or issue restrictions. While VLIWs have demonstrated the possibility of doing this on vector codes, it is unknown whether they can do this usefully on general-purpose codes, especially those with memory access patterns not predictable at compile time.

"Superscalar" processors, in contrast to typical RISC processors, offer rapid execution of multiple instructions with less memory use [20]. "Superpipelined" processors have theoretically similar performance but suffer from penalties such as long pipe flushes and require higher clock rates than the latter architecture. These are both explained in the following paragraphs.

Superscalar processors are capable of fetching, decoding, issuing, executing, and returning results from more than one instruction in a normal stream during each clock

cycle, and thus use the principle of fine-grained parallelism. This can lower the average CPI clocks per instruction from 4-7 for a CISC to 0.3-0.5 as in the i960 [12]. This can be implemented such as in the i960 with a wide instruction bus between the on-chip instruction cache and instruction decoder, allowing multiple instruction words fetch per clock cycle. The i960 can do a register-to-register operation, a memory operation, and a branch operation simultaneously. Normal RISC processors implement a non-interlocked execution pipeline wherein operations utilizing more than one clock cycle may not be immediately followed by instructions requiring the results and Null instructions may be needed. But superscalar processors use "scoreboarding" to automatically detect interdependencies and thus appropriately delay issuing of dependent instructions, and to wait automatically for results to return as needed. This feature contributes to code compactness such as in the i960 with 20-30% smaller code than other RISC processors, a good feature for embedded systems. These features also help improve performance in systems with memory subsystems with multiple wait-states since the processor may be able to continue executing instructions while data is being fetched, unlike RISC processors which require operation stalling until data is available. Thus RISC processors are intended for use with large off-chip caches.

Superscalar processors restrict the combination of instructions issued per clock. For example, a two-way superscalar processor restricts execution to one floating-point and one integer instruction together. In this case the machine would be very ineffective in a program requiring no floating-point operations. Less restricted issue of instructions combinations such as in superpipelined processors or the use of dynamic hardware scheduling can help alleviate these limitations. In fact, studies have shown that machines with lower issue rates can have higher performance if not limited by operation restrictions [14]. In a superscalar machine, some of the functional units may have to be duplicated and the number of register ports may have to be increased for maximal performance. Also, as memory references constitute many of the instructions, multiple data caches may be needed.

A superpipelined processor refers to one with its clock cycle shorter than the length of its shortest operation. The processor clock frequency is increased, the instructions are broken into smaller sub-operations, and these latter are overlapped if possible. The superpipelined unit has no parallel internal execution units but has many pipelined stages operating at a fast rate. These stages may be implemented as several short pipelines or as a few long pipelines with a control unit able to fill several pipeline stages per cycle. Fetch instructions are overlapped, as well as decoding, execution, and storing. When a

conditional branch is pending, scheduling useful instructions is difficult unless the outcome of the branch is known. It has been suggested that initiating computations in anticipation of the outcome of branches may improve performance of such a superpipelined system [19]. Due to the shorter pipeline in superscalar processors, branch operations causing pipe flushing have less impact than in superpipelined processors.

As instruction-level parallelism is often the limiting factor to the speedup in superscalar and superpipelining processors, there is a tradeoff between the depth of a pipeline and the instruction issue rate. That is, if the latency of functional units is made deeper, the instruction issue rate will drop. Thus with a limited silicon budget a choice must be made between bandwidth of a functional unit and its latency. Most emphasis is focused on bandwidth, but designing a unit with lower latency and a lower bandwidth may be preferable [14].

2.2.2.8. Caches

Data caches generally operate in one of two modes: write through or copy back. On a cache hit a write through cache updates the data in the cache and writes the data to main memory via the external bus. A write through cache generally does not allocate on a write miss and so the only way for new data to be placed in the cache is on a read miss. For a write hit to a copy back cache, the system updates the data in the cache but does not update main memory, thus leaving stale or incoherent data in main memory. The processor then only accesses data in the cache, and if other data is needed it is loaded into the cache after having saved the current modified cache contents so as not to have incoherent main memory. Copy back caches generally allocate on a write miss. In designing the MC68040 it was determined that its usage would be mainly in uniprocessor systems; in such cases, the copy back method maximizes performance and minimizes bus usage.

To avoid displacing data in the cache unit while accessing large data structures and in embedded instrumentation controllers which access memory mapped I/O devices, it is useful to have a "load direct to pipeline" instruction which bypasses allocation of cache area. This is referred to as a "non-blocking" cache and allows other memory accesses to occur while a miss is being handled. Conversely, having a locking option for critical code in caches may be necessary in order to keep critical code in the cache.

Some mechanism must be provided to optimize cache misses. Thus, in the i860 a read-miss results in four bus cycles to fill the 32-byte cache line enabling the processor to continue at full speed. A write-miss will be stored into one of two 128-bit write-buffers which store directly to memory, in parallel. Thus, the write-buffers support two store-misses and a delayed write-back of the dirty cache line.

In certain processors, the data cache may be viewed by the user as a series of registers. These are very convenient and fast to access. For example, the MC68030's 256-byte data cache may be viewed as a 64-entry by 32-bit extension to the normal eight data registers. To implement this, the cache must have a freezing mechanism so that the cache controller does not change the cache contents while they are being used as registers.

Cache-miss ratios can be improved by increasing the number of bytes per cache line, where a cache line is the quantum size of data transferred to and from cache memory [19]. A longer cache line prefetches potentially useful data to the cache in addition to the required data. Unfortunately this increases memory traffic.

Cache access time is nearly always the critical path in processor design, determining processor cycle time [14]. Average memory access time may be expressed as the sum of hit time plus miss rate times miss penalty. Thus cache size, associativity and block size will have optimal values to minimize average memory access time. As the gap between processor cycle time and DRAM access time increases, the impact of misses leads to processors losing their performance to the memory system.

Hewlett-Packard research shows that small 4 to 8 KByte caches result in a 25% miss rate in workstation environments and that processing misses in the primary cache create a bottleneck. However, Intel argues that in many applications an 8 KByte cache may suffice and this saves the user from having to implement an external second-level cache. In embedded processors the instruction code is usually linear but the data accessed may be very non-continuous, such as in a postscript graphics processor for a printer where two adjacent pixels may be separated by 1KByte in memory [18].

Caches differ in their level of associativity. The simplest cache is a "direct-mapped" cache, known as 1-way set-associative. A group of contiguous memory locations is referred to as a block of memory. If each block has only one place in can appear in the

cache, the cache is said to be direct mapped. The mapping is usually (block-frame address) modulo (number of blocks in cache). If a block can be placed anywhere in the cache, the cache is said to be "fully associative". If a block can be placed in a restricted set of places in the cache, the cache is said to be set associative. A set is a group of two or more blocks in the cache. A block is first mapped onto a set, and can be placed anywhere within the set. The set is usually chosen by bit selection as (block-frame address) modulo (number of sets in cache). If there are n blocks in a set, the cache placement is called "n-way set associative". Because of their inherent simplicity, direct-mapped caches offer 2-10% faster hit times if externally implemented and negligible hit times when on-chip. They also require a smaller die area than more complex caches. For higher hit-rates, 2-way or even 4-way caches are used [21].

Unlike UNIX programs that switch between large programs, caching real-time embedded applications can result in high hit rates as these programs are often small and have few tasks active simultaneously [12].

2.2.2.9. DSP issues

In addition to the high-speed computation blocks, DSP architects have begun including microcontroller features such as serial and parallel ports, counter-timers, and ADCs and DACs. More analog front-end processing, graphics handling and microprocessor features will appear on future generation DSP chips [22]. Most DSP chips have three basic features, namely a high-speed arithmetic unit with a hardware multiplier for either integer or floating-point operations, RAM, and ROM for program and data storage. The divergence in DSP chip architectures is caused by differences in host-interface support, I/O ports, special features to aid in program execution, and links to real-world signals such as ADCs and DACs, sigma-delta converters, and frequency synthesizers. Some examples are multiple address generators that address data and program memories in parallel in the ADSP-2100 family, a 15-word instruction cache in AT&T's DSP-16 series for short code loops and zero-overhead looping, and complex-number processing in the ST189xx series from SGS-Thompson. National added a DSP block to its general-purpose 32000-family CPU core whereas TI and Microchip added control and microcontroller functions to their DSP core. Motorola's M56100 series is based on a 16-bit DSP core macrocell optimized for voice and data applications. Intended for designers to use similarly to ASIC macrocells for custom DSP creation, it can do six operations simultaneously and

needs only two instruction cycles to start handling an interrupt, enabling it to handle real-time systems.

Math operations such as multiply-and-accumulate MAC are common in DSP algorithms such as filtering. The MC56100 core includes a single-cycle 16 by 16 bit multiplier and dual 40 bit accumulators. The wide accumulators allow less overflows and round-off errors than the normal 36-bit width resulting in a cleaner signal and less background noise [22].

2.2.2.10. Other Design Issues

DMA channels are a common and powerful tool in computer systems. An on-chip DMA can relieve the CPU of burdensome input/ output operations and maximize CPU performance. The main drawback to DMA in a shared bus architecture is that the CPU can be denied access to the bus while the DMA is occurring. Cycle stealing is the normal mode of DMA transfer in such an architecture. Cycle-stealing, in which the DMA processor "steals" bus access time between CPU transactions or, for example, one out of two bus access time slots, does not really affect CPU performance if the CPU's bus occupancy is low. Another mode of operation is the "burst-mode" in which the DMA takes control of the bus until it completes the transfer. Burst-mode disables the CPU's ability to respond to interrupts and this may be undesirable in certain systems. Thus, the DMA controller can have an input interrupt signal to dynamically remove it from burst mode to relinquish the bus. To accommodate lower performance peripherals requiring memory, it is possible to tie these into the DMA over a DMA-to-peripheral port. The DMA can then pack or unpack to the full data bus width to and from the memory as well as to and from the peripherals which commonly use 8-bit accesses as well as prioritize the accessing priority of the peripherals' request for memory accesses.

There are two options in providing semaphoring capabilities. The most common is to provide a locked read-test-write "test-and set" instruction. The second option is to provide a lock and an unlock instruction which provides locked instruction sequences, up to 32 in the i860 processor. Even though it would appear that the i860's lock/unlock mechanism provides better support for such activities as a counting semaphore, the actual number of instructions required to implement such a construct and its execution speed is

approximately the same as for the MC88000 or the SPARC which use the "test and set" method [3].

Although registers have very fast access times, they occupy a large amount of silicon real-estate. Replacing a large set of registers with a single and compact SRAM block will save much chip area. The SRAM requires a longer access time but this may be acceptable in lower performance systems. Furthermore, pipelining can eliminate this additional access time. If multiple simultaneous register accesses are required, such as in binary systems, then a dual port SRAM known as a DPSRAM may be used. This device is an SRAM with two fully independent address and data ports. Its sole restriction is to not perform a write to a location during the same cycle from both access ports. The Siemens 80C167 utilizes a DPSRAM, with a pipelining mechanism to eliminate access time, as register bank, allowing it to offer register windowing at a low cost.

Some of the parameters desirable in a processor are a higher integration for an application including minimizing so-called glue-logic, and more advanced technology be it in architecture or in processing capabilities, expandability, and performance. These must be achieved while meeting a competitive schedule, compatibility with previous processors, and price. Designing functionally independent modules such as CPU, caches, and communications coprocessors which may be mixed and matched such as in the TMS370 family, allows the designer to achieve a modularity for easy expandability, adaptability, and upgradability.

VLSI chips must be able to accommodate a variety of system designs. Thus, they must support the capacitive load of the other IC's in the system, which requires power, and thereforw they have large I/O buffers. Large capacitances slow down chip to chip communications, and so there is a trade off between capacitance support and system speed. In the WE32100 chip set, the no glue-logic requirement meant the chips had to support 130pF loads, limiting the operating frequency to 18 MHz. Inductance can induce on-chip noise voltages and ground bounce which can cause logic errors in the processor [13]. To help keep these induced voltages below the logic threshold levels, multiple power and ground pins may be added to the chip. Also, the chips may be designed to minimize the number of concurrently switching on-chip signals, for example, by asserting the data and the address signals one clock cycle apart. Timing must be guaranteed over best to worse case operating conditions and over processing variations, which is particularly tricky in asynchronous designs.

The first generation of microcontrollers involved integrating serial ports, timers, and other elements on chip. The second generation is now integrating units such as RISC and DSP hardware onto the MCU [23]. A detailed overview of 43 microprocessor and microcontroller chips is given in [24].

Real time detection of performance may be incorporated in future systems in which performance degradation may be corrected by re-allocating resources dynamically by program recompilation during execution [19]. For example the cache may be useless with a particular data structure, or the scheduler may be changed for more optimal execution.

Speculative computing occurs when an operation is executed before it is known whether or not it is needed. This ignores control dependencies and hence makes highly accurate branch predictions critical.

2.2.3. SUMMARY

This section discussed a number of topics for design of an embedded instrumentation processor. Among the topics presented was a review of RISC architectures since RISC provides relevant advantages over traditional CISC architectures. A review of bus interfaces, interrupt requirements, and von Neumann versus Harvard architectures was provided. Some issues relating specifically to embedded processors were then examined. Pipelining, parallelism and cache issues were also discussed. Finally, digital signal processors DSP and other various design issues were examined. These issues along with the issues of chip design methodologies, testability, and processor value to be examined in following chapters, provide the basis for the design of an embedded instrumentation processor.

2.3. CHIP DESIGN METHODOLOGY

2.3.1. INTRODUCTION

Design of a processor is a complex and multi-disciplinary activity. An organized approach to design must be developed and followed for successful implementation. Teams of several hundred people sometimes distributed in several countries are involved in such design tasks and it is important to coordinate these efforts in a single direction. Specialists

in architecture, hardware, software from application to operating system levels, semiconductor technology, and finances must work together for such a design. The prime importance of a final goal of efficient execution of the application software and this effect on the end-user must be maintained throughout a design. Chip design strategies range from top-down hardware design, to compiler creation first followed by hardware design, to copying or "reverse engineering" a previous chip and improving it. Many chip implementation strategies are possible, ranging from fully manual to fully automatic, each extreme having certain advantages. Four design approaches are reviewed in this chapter, namely for the development of the TX1 processor, the MC68040 processor, MIPS processors, and the PA-RISC 1.1 processor. These approaches along with the previously reviewed issue of VLSI design for embedded control and the issues of testability and processor value examined in following chapters, provide the basis for the design of an embedded instrumentation processor, the topic of this thesis.

2.3.2. CASE STUDIES

2.3.2.1. Development of the TX1

The following description relates to the development of the TX1 processor [5]. First, CAD tools were widely used and no breadboarding was done. Logic synthesis and PLA tools were used in the logic design stage and an automatic placement and router was used in the layout stage. Verification tools used were a function simulator, a logic simulator, and a comparator between layout data and logic net data. Second, high-speed operation was obtained using state-of-the-art technologies such as 1 μ m CMOS technology, a one phase clock system, and a technique for reducing clock skew. Third in order to enhance testability three testable design approaches were implemented and optimized, as described in Section 2.4. The logic design was done both with thirty thousand lines of a hierarchical hardware description language used by an automatic design method and also by manual design. The automatic method was used mainly for control blocks. Data path parts such as the ALU, barrel shifter, and most registers were designed manually to achieve both high-speed operation and small chip area. Having the control section computer generated reduces development time since the control section is modified much more frequently than data paths [5]. Layout densities achievable for automatic design is 1.6 kilotransistors per mm² whereas for manual design the layout density achievable is 5.4 kilotransistors per mm², not counting ROM, RAM, and PLA densities for the TX1. The floor plan of this chip was separated into control parts and data-path parts in order to meet chip-size and design-time

requirements. It is shown [5] that with this mixed automatic and manual method of design, a chip size of only 18% greater than full manual method is obtained, whereas the full automatic method would yield a chip size of 53% greater than full manual method. Additionally, design times for full and semi-automatic methods are the same at 48% less than those required for fully manual design methods. This low chip area and low design time is possible for the following reasons. First, regular blocks such as ROMs and PLAs were designed manually and dynamically changing blocks such as the control blocks were designed automatically. Second, functional verification was done in parallel with layout design, but this method suffers when errors are found in the previous design stage which affect the current design stage. It also implies availability of more design team members to work in parallel.

2.3.2.2. Development of the MC68040

This MC68040 processor was designed in a top-down fashion in three blocks which are the Integer Unit IU, the FPU and the memory subsystem [25]. The three sections were subdivided into control stores, random logic, data path elements for the IU and FPU, data and instruction cache/tag memory, address translation caches, internal memory controllers, and the external bus controller. This modular approach divides the processor into parts that exhibit minimum interaction with each other. Then the high level modeling was done, using register transfer language (RTL) mostly in data path sections and using functional descriptions mostly in controller sections. Once debugged, implementation was performed. For the data path sections a transformation of the RTL style model served as a register data path. The more random areas required more automated logic synthesis approaches, where a tool was used to extract the logic in the behavioural models and then feed it to tools such as Espresso and MisII to produce layout ready transistors which were then synthesized directly into layout. Design verification was done in a hierarchical method since each section had different verification requirements and needed to be debugged independently. This approach also allowed the evaluating of design changes and design error fixes as quickly as possible. Two methods provided early verification of the modules and were instruction sequences for the IU and FPU and dedicated stimuli for other modules [25]. At the last stage of the verification process, the entire assembly was simulated with the same instruction and data sequences. This hierarchical verification environment also provided a framework for the verification of the so-called structural or gate level description by replacing a behaviour section with the gate representation and verifying proper module operation with the same stimulus.

2.3.2.3. Development of the MIPS Processors

MIPS built its compilers before going to the microprocessor design, running a wide range of application programs to see where compilers spend most of the time [26]. Hardware was then implemented to make these functions as fast as possible, and many of the other functions traditionally done in hardware were done in software. Instead of using a large silicon real estate with large register files, which reduces register saves and restores on procedure calls, interprocedural register allocation techniques were implemented in compilers to keep slow memory transactions to a minimum.

2.3.2.4. Development of the PA-RISC 1.1

The development of Hewlett-Packard's PA-RISC 1.1 architecture was strongly influenced by the need to balance cost and performance and constitutes an upgrade to the previous PA-RISC architecture. In addition, processor and memory subsystems had to be compact, consume little power, and require minimum support hardware. Inputs from manufacturing, technology partners, VLSI and system designers, architects, compiler and operating system designers, performance analysts, and customers were all considered. Because time to market was a key design focus, many features for this chip set were drawn from the PCX chipset used in HP's high-performance multiprocessor systems but modified to accommodate a lower cost system better suited for the target markets of engineering workstation and low-end multi-user computer systems. For example, the memory system design was changed from a two-way set associative cache requiring VLSI support chips to a built-in interface supporting TTL-I/O SRAM chips. For architectural enhancements, a cross-functional team consisting of people from processor, graphics, compiler, performance analysis, and architecture teams were assembled with the mandate of identifying opportunities for enhancing system performance by instruction-set extensions. Suggested enhancements were validated by analyzing graphics and numerical applications, with hardware and software costs being considered as well.

HP has invested in the development of specialized hardware and software instrumentation to aid in analysis of computer system behaviour under actual customer workloads. This results in the capability to characterize workloads to any level, from operating system call graphs to pipeline interlocks, as well as analysis of CPU, cache,

memory, and I/O behaviour non-intrusively. The resulting data from actual customer workloads is used to drive detailed CPU models which are used to quantify potential performance gains from instruction-level parallelism, specific processor implementation decisions, compiler scheduling algorithms, and operating system policy changes. In the PA-RISC development, this type of information guided decisions related to cache design, pipeline trade-offs, branch prediction policies, and so forth. An example in the influence of this approach is in the design of superscalar machines. Failure to consider customer workloads can lead to exceptionally complex designs with little benefits even though this approach is meant to provide substantial performance improvements. In addition, the complexity of these designs can limit clock rates and future performance scaling. Use of these models has enabled HP to predict performance of multiuser systems within 5% and computational system performance within 2%.

Time to market and manufacturability were enhanced by designing an external primary cache. If placed on the CPU chip this could drive up cost and decrease yields. As well, the fabrication technologies for high-speed logic and high-density memory are different.

While designing the pipeline and deciding how to split the instruction stream between functions and the number of stages involved, the designers chose to optimize for average or sustained performance for rather than peak performance. Sustained performance includes non-ideal events such as pipeline stalls because of code branches or hardware resource contention measured while running real applications.

Rather than implement a superscalar processor, other critical factors were focused upon such as circuit design techniques, clock speeds, cache memories, main memory latency, and so forth. These are far more critical in determining basic competitive attributes such as cost, performance, and time to market. A future growth path for the architecture was built in since a doubling in performance is expected every 12 to 18 months. Some of the major issues for HP's PA-RISC 1.1 were as follows. By using efficient circuit designs to achieve high-speeds rather than pushing technology requirements to their limit, room was left for quick speed increases through technology scaling. Emphasis on improving the performance of packaging and off-chip cache timing paths offers further opportunities for growth as faster SRAM chips become available. Avoiding use of advanced multichip module packaging technology will allow to minimize chip-crossing delays and clock skews by evolving to multichip module based designs. Adopting a multichip design that did not

push the limits of any individual chip density or die size enhanced both manufacturability and time to market. This leaves space for future release to have more features integrated both for cost and performance enhancements. Finally, designing the system for external cache memory sizes 4 to 8 times larger than those presently used will allow use of new SRAM density and cost offerings [27].

2.3.3. SUMMARY

Four chip design approaches were reviewed in this chapter. It was seen that usage of automatic synthesis techniques for design of control logic, and manual synthesis techniques for design of repeatable structures is the best compromise for minimal chip area usage and minimal development time. Modular, top-down designs are testable and control design complexity. Extensive analysis of targeted application software and of application hardware requirements leads to useful and efficient designs. Using sophisticated compiler software maximizes hardware simplicity and efficiency. The cost of design features versus their actual benefits must be balanced for a successful implementation. Usage of known technology and implementation of expandable architectures give predictable, flexible, and cost efficient designs.

These issues along with the previously reviewed issues of VLSI design for embedded control, the issues of chip testability and of processor value to be examined in following chapters provide the basis for the design of an embedded instrumentation processor.

2.4. TESTABILITY ISSUES

2.4.1. INTRODUCTION

It is important to "design for testability" from the beginning of a processor design and several approaches or combinations are possible. Some major aspects of this important topic are discussed in this section. As a rule of thumb, the effort required to develop test programs grows in relation to the the cube of the number of equivalent logic gates on the chip [28]. The purpose of design for testability is to simplify testing of the chip by increasing the visibility of the chip's internal circuitry by breaking it into simpler smaller functional blocks and then providing ways to stimulate these functional blocks directly with test patterns and examine the resulting outputs. It is also possible to include on-chip

circuitry to generate the test patterns that stimulate the functional blocks and capture the resulting outputs in a compressed format. A more sophisticated approach is to include diagnostic logic on the chip to evaluate whether the tested vectors were those expected. This is referred to as a chip with full Built-In Self Test (BIST) capability. This concept can even be extended to provide support for board level in-circuit testing. The relevance of controllability, observability, and predictability are discussed. These issues, along with the previously reviewed issues of VLSI design for embedded control, chip design methodology, and the issue of processor value are examined in the following chapter and provide the basis for the design of an embedded instrumentation processor.

2.4.2. SCAN METHODOLOGIES

The main aspects of BIST are "controllability", "observability", and "predictability". The latter refers to how well the initial state of the processor is known, since an initial uncertainty implies difficulties in synchronizing test programs with the chip under test. A common way to ensure start-up predictability is to use a RESET sequence. Controllability refers to the flexibility allowed to control the inputs to embedded logic, since this is necessary to apply the test patterns. Observability is the degree of access included so as to see the internal response of the circuit to the applied test pattern. The circuit is partitioned into "subnetworks", at the boundaries of which test circuits are placed. These must allow both the normal subnetwork inputs to be switched to the test-pattern inputs, and the outputs to be viewed externally. The degree of partitioning is important as too much will require excessive test circuitry and not enough provides little information. Subnetworks should contain little sequential circuitry; they should contain only combinatorial circuitry and if possible they should break up feedback loops.

Scan-In Scan-Out designs allow access to these functional blocks without greatly increasing the number of pins on the chip and the subsequent cost. Inserting multi-purpose latches between functional blocks and chaining them up as a serial input and serial output, allows loading in stimulus data for the next block using a single wire and allows unloading data from the last functional block through a single pin.

Such a system is implemented in IBM's Level Sensitive Scan Design LSSD. In LSSD all storage elements are realized as latches which are level sensitive devices rather than flip-flops which are control edge triggered devices. The level sensitivity of latches can ease

timing restraints on inputs since these have the entire clock pulse to settle. It also makes latch operation more insensitive to degradations of the edges of the clock waveforms. LSSD also requires all latches to be cascaded into a long shift register to load and unload the test patterns. This is done by using a standard module known as the Shift Register Latch SRL and uses two clocks in addition to the normal system clock. See Figure 8 for details. The SRLs allow both test mode operation and normal pass-through operation. Refer to [28] for details. Published data shows that LSSD implementations yield a four to twenty percent increase in circuit complexity [28].

2.4.3. SELF TEST AND DIAGNOSTIC METHODOLOGIES

In addition to incorporating scan methodologies, the processor may include methods to perform tests on itself, generating its own test vectors. Test vector storage would require megabytes of memory and thus an algorithmic test pattern generator is required. As well, the expected outputs should be compacted to reduce storage requirements. The simplest method to generate the test vectors is exhaustive testing where all possible inputs are generated. This is feasible only on small logic blocks. A pseudo-random test sequence may be used, but this requires knowledge of the length of a sequence needed to have sufficient fault coverage.

Circuits incorporating sequential elements in which a random sequence could place the circuit in a frozen state early in the test sequence could be fatal to this type of testing. "Signature analysis" is a method in which, knowing the starting state and the number of clock cycles, a block may be cycled through many states and only one in particular need be checked. A linear feedback shift register LFSR may be used to generate a pseudorandom binary sequence as a Pseudo-Random Pattern Generator PRPG [28].

Figure 8, Structure of the SRL and Its Use in an LSSD Network [28]

The LFSR is simply a multibit shift register with the output of some of the stages fed back into the input through exclusive OR gates, the position of the feedback points determining the length and composition of the sequence, with a maximum length for an N stage LFSR being $2^N - 1$. An LFSR in which the feedback bits are combined with an output bit from the circuit under test achieves compaction of output data, since any feedback change will result in a different and exclusive final state in the LFSR. This is referred to as a Multiple-Input Signature Register (MISR). Thus, if the sequence of data from the block being tested is known, then the final expected state of the LFSR may be computed in advance and is called the signature. Another configuration for both the random input generation and the output data compression is called a Built-In Digital Circuit Observer, or Bidco. A built-in logic block analyzer, or Bilbo, is a standard structure used to implement either a PRPG or an MISR and may be operated in several modes, namely ordinary parallel load register, pseudorandom number generator, or parallel signature generator. Refer to Figure 9.

In the MC6804P2 single chip computer, the additional ROM needed for the BIST is 288 bytes and additional chip area for the BIST is less than five percent with a fault coverage of greater than 99% of possible single stuck at faults [28]. The TX1 processor [5] uses three methods for testable design: parallel scan test design, macro block test design, and self-test. The scan test is used to test the sequential logic of the processor. In the TX1 the 1660 flip-flops are divided into 30 groups and connected to each other in order to realize 80 scan paths. The macro block test enables independent tests for the ALU and barrel shifter blocks. The macro blocks can be controlled and observed through a 32-bit data bus. Self-test performing signature analysis and data compression is widely implemented under microprogram control and runs at the same speed as normal operations. The fault coverage achieved exceeds ninety percent and the additional area for including testability is 4.6 percent of the chip area.

2.4.4. SUMMARY

This section covered the issue relating to "design for testability", and the importance of including this from the beginning of a processor design. The purpose of design for testability is to simplify testing of the chip by increasing the visibility of the chip's internal circuitry by breaking it into simpler smaller functional blocks, and then providing ways to stimulate them with test patterns and to examine the resulting responses.

One stage of a Bilbo

Eight-Bit Bilbo Schematic

Figure 9, "Eight Bit Bilbo Schematic", Adapted From [28]

It is also possible to include on-chip circuitry to generate the test patterns that stimulate the functional blocks and capture the resulting outputs in a compressed format. By including diagnostic logic on the chip, it can self-evaluate whether the tested vectors are those expected. This is referred to as a chip with full Built-In Self Test BIST capability. This concept can even be extended to provide support for board level in-circuit testing. The relevance of controllability, observability, and predictability have been discussed. These issues, along with those of VLSI design for embedded control, chip design methodology, and processor value, to be examined in following chapter provide the basis for the design of an embedded instrumentation processor.

2.5. MEASURING PERFORMANCE OF A PROCESSOR

2.5.1. INTRODUCTION

Both during and after design, it is important to determine the performance measure of a processor. Performance can have different meanings to different specialists exercising different functions. Although raw performance of a processor is of prime importance, other issues, such as cost, functionality, and ability to execute the intended software in the intended environment at the intended performance level, are also very important. Sections 2.2 and 2.4 discussed detailed issues relating to architecture and testability. These factors all contribute to determining the “value” of a given processor. This section is divided into two main sections, the first dealing with measurement of performance of a processor.

Performance is difficult to define and measure and is a controversial topic. Much effort has been devoted to resolving this, and three solutions are discussed in the section on performance. The main problem stems from the need to compare different processors with different architectures and different functionality on an equal footing in terms that each specialist will be able to relate to. Metrics are used to evaluate and compare different designs from different points of view. For example, CPU performance may be defined by measuring execution time on a workload, where execution time is the product of the number of instructions required by the average number of clock cycles per instruction by the time per clock cycle [14]. The issue of performance and chip technology is introduced. The second section in this chapter discusses other factors affecting value of a processor such as cost, functionality, and ability to execute the intended software in the target environment at the intended performance level. These issues along with those of VLSI

design for embedded control, chip design methodology, and testability provide the basis for the design of an embedded instrumentation processor.

2.5.2. MEASURING PROCESSOR PERFORMANCE

2.5.2.1. Method Number One and Its Analysis

An extensive model of processor performance is given in [26, p.56], but source [29] suggests a simple model of processor performance may be expressed as

$$\text{Performance} = (K1 \cdot t1 + K2 \cdot t2)^{-1}$$

Where K1 [microcycles/operation] is the number of microcycles per machine instruction, t1 [seconds/microcycle] is the data path cycle time, K2 [bits/operation] is the memory used per operation, t2 [seconds/bit] is the memory access time.

Ways to decrease K1 are given as having multiple registers or register sets, having multiple data paths, having multiple function units, allowing processor/memory overlaps, having fewer microsteps (microinstruction decoding, more parallelism), having multiplexing processor logic (for example, a barrel shifter), having a more efficient Instruction Scheduling Processor, or tailoring microcode flow according to macroinstruction usage frequency. Ways to decrease t1 are given as using a faster technology (for example, for whole data path or only critical components), shorter microcycles (for example, multiple length microcycles), and more efficient microinstruction fetch (for example, interleaved control stores, multiple microword fetch, pipeline microword fetch/execution). Ways to decrease K2 are given as to increase operand bits/instruction (for example, scalar versus serial data types, vector versus scalar data types), having a more efficient Instruction Scheduling Processor. Ways to decrease t2 are given as utilization of a faster technology, building in an apparent speedup (for example, I/O spaces, caches), widening word access (for example, making the data path wider, multiple fetches on a multiplexed bus), and a more efficient utilization of bandwidth (for example, instruction prefetch and processor/ memory overlap)

2.5.2.2. Method Number Two With an Example

For any benchmark, the performance P of a particular computer is inversely proportional to the product of the benchmark's instruction count I , the average number of clock cycles per instruction C , and the inverse of the clock speed S [4]:

$$P \propto (I \cdot C \cdot 1/S)^{-1}$$

Following is a comparison using this method of RISC and CISC machines. Assume the same clock speed S , a number of clock cycles per instruction C of 1.3 to 1.7 for RISC processors, and 4 to 10 for the CISC machine. Thus the instruction execution rate is $4/1.3=3$ to $10/1.7=6$ times faster for the RISC than the CISC. But because 20 to 40% more RISC instructions are required to perform the same programs, it may be concluded that a RISC machine is 2 to 5 times faster than a CISC machine for typical large programs [4]. RISC code is not actually as large as expected, due, for example, to the availability of many registers which simplify context switches and thus reduce code size. Thus RISC chips dramatically reduce clock cycles per instruction and slightly increase instruction count per program, achieving an overall performance increase.

2.5.2.3. Method Number Three With an Example

Performance can be measured in terms of functions performed per time unit or conversely as the time needed to perform a particular task. There are two basic performance measurement methods, the first is probabilistic which takes into account process variations, concurrent activity in the system, and the system design both in hardware and in software. The second method is deterministic, in which it is attempted to remove variations by assuming worse case, average case or weighted average statistics. Different methods of measuring performance in a deterministic way include measuring typical instruction time, memory size, or using Kiviat graphs. The Kiviat graph, as shown in figure 10, summarizes major performance parameters and shows the balance between different parameters of the system in a qualitative way. According to G. Amdahl [29] while designing the IBM System/360, there are two such balances to keep. The first rule relates P_c , the processor instruction execution speed, to M_p , the memory size by stating that 1 byte of M_p is needed to support each P_c instruction per second. The second rule relates P_c speed to I/O Bandwidth by stating that 1 bit of I/O is required to support each P_c instruction per second.

Figure 10, Sample Kiviat Graph [29]

2.5.2.4. Performance and Chip Technology

The number of components per chip of a given area is $O(n^2)$ function of feature size. Circuit delay is roughly proportional to feature size and is inversely proportional to speed, but this delay is limited at higher frequencies by interconnect delays. By multiplying the number of devices by their speed, lithography is responsible for $O(n^3)$ improvement in computing capability. Since the number of components increases as a square law while their performance increases only linearly, effective utilization of the components is critical. As well, chip area increases have an $O(n^2)$ effect on the number of devices available. However this does not change basic device performance and thus has less impact than reduced feature size. The easiest way to use the $O(n^2)$ number of devices available due to chip area increase is to use them as memory cells. This has resulted in on-chip caches which reduce inter-chip communication delays and result in a hierarchical cache system.[14]

2.5.3. OTHER FACTORS AFFECTING VALUE

In addition to performance, the value of a processor depends on many other factors. The cost of the chip must be balanced with its performance and functionality within an actual target system. For example, even though the performance of a chip with a wide data bus should be superior to that of a chip with a narrow data bus, the latter allows a lower cost in implementation of the entire system. This may be a primary goal in the embedded control system being designed. The fit of the processor to its designed target environment is extremely important, such as its suitability to the application software. For example, a chip with register windowing running C code is more apt to respond quickly to interrupts with a low latency rather than a system such as a CISC which typically must copy all internal registers out to memory before being able to begin response to an interrupt. Having many registers also suits a high level language such as C better than having few registers. The cost of a chip is both proportional to chip area and to pinout size. Thus, it is important to carefully balance how much functionality is required with the actual implementation value per feature. Typically, more functionality means more chip area and more pinouts, which means more cost. For example, many chip families use a separate math coprocessor chip in order to reduce processor chip area and to allow the customer to choose whether or not to incur the extra cost of the added functionality.

Another example is HP's decision in implementing the PA-RISC 1.1 architecture. The processor and primary cache were implemented as separate chips for reasons of time to market and manufacturability. In fact, in this case another side benefit occurred by removing the memory from the main processor. Fabrication techniques better suited to memory were used for the cache chip, thus potentially lowering chip cost and increasing its value. Appropriate peripherals must be provided in order to make the chip attractive. For a data acquisition and control system, typical peripherals include ADCs, DACs, a pulse width modulator, digital I/O, timers, serials ports, and even highly functional units such as the time processor unit TPU on the MC68332, a RISC coprocessor for complex real-time control. Examples of this may be found in [24]. Incorporating the board's glue-logic onto the processor simplifies board design and lowers its cost. For example, the MC68332 includes an on-chip clock driver, clock multiplier and clock divider for generating system and serial baud rate clocks from an external crystal. It also has a very flexible system integration module which serves as bus between chip modules and allows software redefinition of the chip's pinout functions for custom functionality tailoring. Secondary considerations affecting value of a chip include support for fault tolerance, power usage, and power management features. Other factors such as chip size, heat dissipation and cooling requirements also affect chip value.

2.5.4. SUMMARY

This section covered determination of performance measurement of a processor. Performance can have different meanings to different people exercising different functions. Although raw performance of a processor is of prime importance, other issues such as cost, functionality, and ability to execute the intended software in the intended environment at the intended performance level are also very important. Sections 2.2 and 2.4 discussed detailed issues relating to architecture and testability. These factors all contribute to determining the "value" of a given processor. This section was divided into two main sections, the first dealing with measurement of performance of a processor. Three methodologies for measurement of performance were presented to evaluate and compare different designs from different points of view. Even though it is difficult to define performance, it may be said that performance is the number of functions performed per time period. The two ways of observing performance are probabilistic and deterministic, these were explored in this section. The issue of performance and chip technology was introduced. The second section in this chapter discussed other factors affecting the value of a processor such as cost, functionality, and ability to execute the intended software in the

intended environment at the intended performance level. These issues along with the previously reviewed issues of VLSI design for embedded control, chip design methodology, and testability provide the basis for the design of an embedded instrumentation processor.

2.6. SUMMARY AND CONCLUSIONS

Specifications for the embedded instrument processor were detailed in section 2.1. In order to successfully implement such a processor, a literature review on embedded instrumentation processors was presented in section 2.2 to 2.5 as follows. Section 2.2 begun with an introduction of selected relevant topics in embedded instrumentation processor design including a review of RISC architectural highlights especially with respect to traditional CISC architectures. Other issues such as bus interfaces, interrupt requirements, parallelism, caches, and topics specific to embedded processors were covered. Digital signal processors and other various design issues were also examined. Section 2.3 examined a variety of chip design methodologies by reviewing case studies for four processors. The topic of chip testability was discussed in section 2.4 with an overview of design for testability, LSSD scan-chains, and Built-In Self Test BIST. Section 2.5 examined the issue of measurement of the value of a chip, where chip performance measurement is discussed as well as other issues affecting the value of a chip such as cost, suitability to an application, and functionality. The next two chapters describe design and verification of a processor meeting design specifications as per section 2.1.

RISC architectures are optimal for embedded instrument control applications, due to their low usage of chip real-estate, rapid response to interrupts, and the parallel execution of internal units. The saved area may then be used to implement specific "smart" hardware units such as a coprocessor driven I/O. Small data and instruction caches are sufficient for embedded instrumentation software. DSP capability is desirable for activities such as data filtering but not essential. Using a large DPSRAM to implement registers has a higher memory density than discrete register implementations. The saved area may then be used to implement numerous windowing DPSRAM registers for high-level language support with the effect of low interrupt response latency. Together, the smart modules and the large set of registers allow real-time response and reduce requirements for processor core capability.

Extensive analysis of actual application software code and of hardware requirements leads to design optimization. Benefits such as cost and time-to-market of implementing specific features on-chip versus implementing them in software or on a separate chip must be considered. The design must be modular, testable, and preferably include Built In Self-Test features. Performance metrics of the chip may be quantitative, measuring parameters such as data path cycle time and memory access time or using benchmarking programs, or qualitative such as plotting a Kiviat graph.

CHAPTER 3. DESIGN ARCHITECTURE

3.1. INTRODUCTION TO THE DESIGN

The previous four chapters have dealt with issues regarding design of embedded instrumentation processors. First, a description of the application system referred to as the **BarTester** was provided and specifications for the design listed. Then design topics for embedded instrumentation processors in which issues such as RISC technology, bus interfaces, architectures, parallelism, caches, and pipelining were examined. Chip design methodologies were discussed and examples provided. The important issue of testability was overviewed, and finally the necessity of and methods for measuring the value of a processor were examined. These all provide the basis for the design of an embedded instrumentation processor, the goal of this thesis.

The application for this thesis is defined as the design of an instrumentation processor core suitable for used as core in the so-called **BarTester** system, and compatible with addition of on-chip peripheral digital units such as a camera interface and digital I/O. In addition, this core must provide at least equal performance to that of the original system, be of low cost, and integrate as much of the glue-logic as possible. It would also be desirable to include some on-chip memory. Compatibility with existing software would reduce software development costs for transfer of the driver programs.

Several microcontrollers such as those overviewed in [24] have many of the required features. For example, an MC68332 has a modular internal design, is compatible with existing BarTester software, has digital I/O controlled by a "smart" RISC-style coprocessor which could emulate the microsequencer, integrates much of the required glue-logic, has 2 KBytes of RAM, and provides higher performance than the original system. It does not, however, provide enough digital I/O, nor include custom circuitry such as the custom camera interface or a GPIB port. A cost analysis is necessary to balance benefits of developing a fully customized chip, such as presented in this thesis, versus designing a circuit card built around a purchased microcontroller. The focus of this thesis is both on gate level development of and requirements for a low-cost architecture for integrated embedded instrumentation processors. With these features in mind, a processor for instrument control was designed and simulated in order to achieve the design goals. The following chapter first overviews an ideal design, then overviews the design actually implemented and evaluates it. In the next chapter, a detailed description of the actual design is presented which covers the processor piece wise by module and discusses the opcodes it can execute. Then simulation of the design is covered. Finally, chip layout is introduced and an estimate of required layout area for physical implementation is performed and evaluated. This completes presentation of the design, having started from the ideal design derived from theory described in the initial chapters, translated into a practical design, then designed and simulated in detail, including estimation of the required chip area.

3.2. DESIGN DESCRIPTION

3.2.1. Ideal Design

As described in chapters 1 and 2.1, this design is specific to a particular application system referred to as the BarTester. A brief review of the BarTester follows here with reference to Figure 11. Objectives of the design were defined to integrate the **Main Processor Boards** and the **Auxilliary Analog Micro-Control Digital Board** into a single chip. These boards include an MC68000 microprocessor running at 10 MHz with an 8 KByte RAM and an 8 KByte EPROM. The microprocessor controls all system functions, including a GPIB communications link, a parallel interface and timer, a custom camera interface, a microsequencer, an analog bus decoder, and glue-logic. Main design goals were stated to be cost and integration of as much of the system's digital and microprocessor circuitry onto this chip as possible in order to incur a low system cost.

Figure 11, BarTester System Architecture

Some on-chip memory for data storage was described as a desirable feature as well as a means of obtaining architecture that would allow easy integration of custom on-chip peripherals. The chip's performance must be at least of the same order of magnitude as the previous discrete implementation.

The system's requirements will now be examined in relation to the subtopics presented in the chapter "Selected Design Issues". Designing the chip as a RISC processor presents many of the RISC advantages presented earlier. These include a lower design time and chip area usage which lowers the hardware cost. On the other hand, RISC requires more effort for development of a compiler in order to utilize the processor properly, since complexity is moved from the hardware in CISC onto the compiler in RISC.

Single cycle operation and fixed instruction length would allow real-time control with low and predictable latency for interrupt response. A small and register based instruction set would reduce hardware design time, and chip area, and would detach the external bus I/O from internal operations which simplifies internal parallelism design. This leaves real-estate available for implementation of on-chip caches or custom I/O units. Although the RISC code would require a larger external memory it would not be a concern because of the low cost of memory. This would also allow upgrading the system in the future as cheaper and bigger memory chips become available. Register windowing would allow fast context switching for interrupt handling and the memory available by the register windowing could be used as small on-chip instruction caches. A small instruction cache could be sufficient since data acquisition programs generally operate within small loops of code. As explained in chapter 2.2, usage of a DPSRAM for register emulation instead of flip-flop based registers would reduce chip real-estate. The freed up area could be used to expand the DPSRAM in order to allow register windowing or use as on-chip data storage.

A structure allowing fine-grained parallelism would be preferable, with sub-modules executing code at an effective speed higher than that of a single stream machine. This implies that overlapping data or instruction fetch with instruction execution, possibly with a cache, would be beneficial. An internal structure where the instructions are fetched into a small read-only instruction cache would allow storing of tight loops on chip for maximal speed of execution. It would not be very beneficial to have a data cache since the data is read multiple times from the same memory-mapped data acquisition I/O address

whose data changes constantly. This acquired data is then sent directly or with little processing to the analysis computer. Although it would be advantageous to keep this data on-chip with the required GPIB link to the analysis computer, chip-area and pinout considerations suggest it would probably be better to have some form of DMA transferring the data from the external memory to the internal GPIB driver module. This is because this application allows time to transfer the data back to the main computer due to the low-duty cycle the devices under test require. Separate and independently "smart" modules and I/O would off-load the processor from trivial operations and this would allow high-speed system operation even with a fairly low-performance low-cost processor. On-board smart modules for this system include the GPIB interface, the camera interface, a parallel interface and timer, and a hardware control unit with microsequencer, but these are not designed in this project. Internal register and code compatibility with the original MC68000 code would be very desirable as this would allow usage of the original software and represent a major cost savings in implementing the system.

Mechanisms such as good branch prediction could help accelerate code execution, but are not critical in this application. Recall that it is necessary to use a 0.1% duty cycle pulse train while testing the laser chips in this system. Thus, if a 5 microsecond pulse is applied to the laser under test, this leaves 5 milliseconds for data acquisition and transmission of the data to the analysis computer, adequate of time to do several minor computations. For this same reason, DMA channels are not necessary, except possibly for data transmission to the main computer as discussed above. A standard von Neumann architecture with asynchronous bus transfers and a non-multiplexed address/data bus offers minimal pin count and maximal flexibility. The availability of two internal buses could allow simultaneous access to both operands of binary operations. This would be a flexible architecture and would possibly allow the use of both buses as source for the external address and data buses.

An important consideration is to incorporate as much of the system glue logic on-chip as possible in order to reduce external components. It is also important to minimize pin-count which is affected by having on-chip address decoding, for example. As well, general purpose external buses and control lines allowing for future expandability would be desirable for maximum flexibility. As the data is sent to another computer for analysis, it is not necessary to implement units such as a hardware multiplier, but if available such a unit could be useful in scaling and calibrating digitized values, or even for data filtering. Much of the data preprocessing can actually be done in the integer domain with ADD and shift

operations only. It is important to have an architecture which allows expandability, both internal and external. External expandability means that other functional units may be added to the system, and internal expandability means that custom units such as timing modules easily fit into the system. The design must be modular for easy incorporation of LSSD test circuitry to each module. This test circuitry could be combined with a debugging mode for system development which would be accessible to the user. Finally, a low chip area would increase fabrication yields and lower cost.

Thus, for the instrument control application defined, the design's main features are:

- RISC architecture
- DPSRAM based with many registers or register windowing
- Two operand internal bus
- General purpose computation
- Code compatible with MC68000
- Non-multiplexed asynchronous bus, von Neumann type
- Rapid interrupt processing with low-latency
- Custom I/O with "smart" modules
- Fine-grained pipelining
- Small data cache with DMA GPIB interface
- Small instruction cache
- Expandable internally and externally
- Minimal pinout
- Modular and accessible by LSSD, supports a debugging mode.

3.2.2. Actual Design with Performance Evaluation

The actual design description follows in three sections. First, a block diagram for the proposed design is presented. Second, a conceptual overview describes the transition from the ideal system of chapter 3.2.1 into the actual design and is accompanied by a more detailed system architecture diagram. Third, the detailed design is presented at the module level to the reader accompanied by a detailed architectural diagram with control signals.

Figure 12, System Architecture Block Diagram

As shown in Figure 12, the design has three internal system buses and three corresponding external system buses. The internal buses are a control bus, data BUS A, and data BUS B. The control bus contains signals to synchronize system activities and data flow within the system. BUS A and BUS B are data transfer buses. Two buses are used rather than a single one in order to simultaneously manipulate binary data such as two operands to the ALU, or to generate data and address signals for external device access. The external buses are extensions to the internal buses. The external data and address buses are standard microprocessor style data and address buses. The availability of two internal buses allows the external buses to be implemented as an extension of these internal buses. The external control lines provide asynchronous interfacing to external devices. The five internal blocks shown are an ALU, a Microcontroller, a Dual Port RAM, a System Bus Controller, and Other Internal Units.

These will be described in detail in the following sections, but a brief overview is provided here. The Microcontroller controls and synchronizes all operations within the chip. The Dual Port RAM provides memory facility for the chip. The ALU provides simple integer math and boolean operation capability to the chip. The System Bus Controller synchronizes bus activity in order to interface to slow external devices and provides bus contention control within the system. Finally, the block denoted Other Internal Units block denotes the open architecture of the system and is intended to represent application specific hardware such as a camera interface, parallel I/O, timing units, or communications ports.

The primary goal of this design was to design a fully functional processor core for instrument control at minimal cost. Thus, it was decided to build a quasi-MC68000 processor in order to execute the software code almost directly. The original BarTester software designed to execute on a MC68000 could thus be ported with minor modifications only. This dictated the minimal number of internal registers, external buses, internal functionality, and machine opcodes. Another major design issue was the integration level provided by this chip. Glue-logic integration is provided by incorporating address decoding and handshaking decoding on-chip. More importantly, integration capability is provided by having low area and an open architecture, and this chip allows for the addition of both internal and external units. This is possible because of a system bus controller which controls bus usage and clock control for the chip. Thus, it is possible to easily add the

required internal units such as the camera interface, GPIB interface, parallel I/O and timer into the chip's structure, although implementing these units is not part of this project.

The design feature of having on-chip memory for data storage is provided. In fact, the processor's registers are implemented as part of a two-port memory DPSRAM, and the remaining memory locations are available for program access. Using a DPSRAM as register storage facility minimizes chip real-estate. With minor modifications, it would even be possible to use this additional memory for register windowing, allowing fast interrupt response. Interrupt capability has not been provided but is easily added with minimal hardware and minor microcode upgrades. The upgrades to the microcode are to check the interrupt signals' status before fetching each instruction, and if an interrupt is present to call an interrupt servicing procedure. The internal structure has two main 16-bit buses, BUS A and BUS B. These are used internally for binary operations such as simultaneous transfer of data from both ports of the DPSRAM to the ALU's two inputs for operations.

The application code was analyzed and a minimal set of 26 opcodes/addressing modes listed in Table 4 in chapter 4 was found to be sufficient for easy software portability to this processor. This is a RISC-like quantity of opcodes with 92% of the opcodes following RISC-type structure regarding memory-to-register and register-to-register access. One of the two remaining opcodes, or 4%, is very application specific for data acquisition type environments. Using a RISC type approach such as this shortened design time and reduced silicon real estate necessary for implementation. All instructions are one word long, except for long branches that have the offset in a second word. Internal processing is divided into three phases, namely "fetch operands", "process data", and "store result". This takes 2 to 3 clock cycles in the final microcode implementation, depending on actual clock speed. External memory accesses are two clock cycles for a read, namely output address and read data, store in DPSRAM or the reverse for a write operation, namely fetch from DPSRAM, output address and write data. Wait states may be inserted as required by the external devices. To enable proper read and write timing of the internal DPSRAM, a higher frequency external clock is subdivided into three phases used internally solely for the DPSRAM access. Three of these sub-clocks form a single clock cycle for the microcode sequencer clock. It can be estimated that as all operations except external memory accesses execute within three clock cycles, a minimal sequence, the performance of the processor should be at least as good as the equivalent MC68000 performance at the same clock rate. Careful examination of the microcode shows it to be

non-optimized with more than the above described clock counts, but this is simply a matter of microcode tuning, not a design limitation. The effective clock rate given for this design is 16.7 MHz after processing for the DPSRAM access sub-clocks from the external clock of 25 MHz. The designed processor has 16-bit structures, and has an equivalent instruction set as the MC68000 processor for the given instrumentation application. High I/O rates with enough bits is to be provided by an I/O smart co-processor which fits into the open architecture. Thus, it can be concluded that the designed processor core will provide at least equal performance for the original BarTester application which was executing at a 10MHz clock rate on an MC68000 processor. This processor therefore meets design specifications.

In this implementation, no major structuring for internal parallelism was made, although the system bus controller could sequence bus access between parallel units. The internal DPSRAM could be used as a small instructions cache for storing the small data acquisition loops, with less than 50 bytes needed in typical loops. Having a small data cache for packet collection before transmission over a built-in GPIB interface would be beneficial but would require more chip area, more hardware and software design, and most importantly, 20 more pins for GPIB pinout. It would be necessary to do a detailed cost/benefit analysis in this case to determine whether this is an appropriate feature to incorporate on-chip. The design is modular, allowing easy addition of LSSD type edge scanning for complete chip testing. The pinout for the processor is defined in Table 1, with the possibility of modifying actual pinout to either limit the number of pins or to accommodate additional internal units such as a parallel I/O port.

Function	Number of Pins
Power (Vdd, Vss)	2
Data Bus (D15-D0)	16
Address Bus (A15-A0)	16
Control (RES,CLK,RW,BW,DS)	5
Address Decoding (CSi,DTACKi) (i = 1,2,3,5,6,7)	12 (depending on how many are provided externally)
Total	51

Table 1, Pinout of Processor

Figure 13 , System Architecture Overview

The following provides an overview of the actual system design. The system is divided into six internal blocks, as follows. (Please refer to Figure 13 for the internal architecture discussion.) The control section denoted MPC generates control signals to sequence the whole system. The Dual-Port Static Random Access Memory DPSRAM denoted RIM & RAM provides internal register emulation including program counter PC and stack pointer SP as well as a scratch pad memory, which could possibly be used in a windowing register scheme. The Arithmetic and Logic Unit denoted ALU & AIF provides boolean and integer math operation processing.

An Instruction Register IR with pre-instruction decoding denoted BRA provides a register to store the instruction currently being processed and pre-processes this instruction for simplification of the microcode. For example, it compares the current instruction being processed with the current flag status and determines whether branching should occur. It also recodes the instructions so that they fit into a 4 bit space convenient for the microcode to process efficiently. The system bus controller denoted DVI implements address decoding and handshaking with external and internal units, provides external bus interfacing, and can arbitrate internal processing, even placing the processor on hold if necessary. Finally, provision is made for other custom internal units, denoted by Other Internal Units. These could be any circuit necessary for special purposes, a co-processor, or even analog/digital converters. Together these form the entire processor. All internal and external buses are 16-bits wide. This bus width was chosen as compromise between performance and chip area, especially since the analog hardware generates 16-bit digitizing. The internal structure is focused around two 16-bit buses, BUS A and BUS B. BUS A is linked to the external data bus and BUS B to the address bus, via buffers and registers. The clock cycle time of the processor is currently set to 15MHz, mainly limited by the internal DPSRAM's timing which is shown in Table 12 in Chapter 4. It was necessary to implement two further blocks for simulation of the system, namely an external memory source as source of instructions to execute, EPROM, and a clock and reset generator, CKC.

CHAPTER 4. DETAILED DESIGN AND VERIFICATION

4.1. COMPUTER DESIGN ENVIRONMENT

Several design environments are available for IC synthesis. These include, for example, CADENCE or VHDL environments operating on Unix workstations. Due to availability of systems to the author, a mainframe with IBM CAD-terminal based system was used. Northern Telecom device models for the process with internal designation CMOS4 were used. This is not the current mainline technology, since better implementations of CMOS are now available, as well as newer technologies such as BatMOS, a version of BiCMOS which combines bipolar and CMOS transistors with several enhancements specific to telecommunication applications. The CMOS4 and BiCMOS processes are available via the Canadian Microelectronics Corporation CMC.

The tools described in the following section, proprietary to Northern Telecom, with some available for external purchase, were those used by the author. Two levels of abstraction were used. The primary one used predefined circuit cells from a Standard Cell User-Designed Silicon SUDS library. This is a library of predefined and fully characterized components available to the designer. It avoids building a system from the transistor level up and is similar to designing a circuit board using discrete CMOS IC's. The second level of abstraction used was functional behaviour modelling and synthesizes models from these behaviour models. The Functional Modelling Language FML2 was used to describe asynchronous circuits and compiled using FML2C to create a behavioural model of the

intended circuit, including full timing description. Once the model had been simulated, it had to be replaced with circuit components for actual implementation. No synthesis package was available on the mainframe package for automatic asynchronous circuit generation, so manual synthesis was done.

For synchronous system synthesis a package called SYNFUL was used. SYNFUL synthesizes a circuit as either a ROM, OR-AND PLA, or Sea-of-Gates from a state-machine description described using the FML2 language. This is rather like programming, with variables being signals or registers. To design a block using circuit diagrams, a multi-level CAD tool called Functional Description FUNDES is used. The schematics may be compiled by FUNDES into a form suitable for simulation. Simulation for all circuits was done with a package called FUNctional SIMulator FUNSIM. The user may define a series of test vectors to drive the system being simulated and program the expected responses with their timing information in a language called HOLIXZ. After simulation, results may be seen on a text or a graphic terminal using SIMPOST and graphic plots may be generated using SIMPLOT. SIMPOST will highlight simulation errors and warnings as failed by the HOLIXZ file. FUNDES can also perform fault simulation as explained later in chapter 4.3.2.4. The software package PROOFS was used to review results of the fault simulation. CLASSIC is a program used for synthesis of RAMs such as the DPSRAM used in this design, which creates timing diagrams, models for simulation, and layout of the RAM synthesized. After simulation, the circuit is laid out using computer assistance. Programs available include AUTOLAY2. The layout may be analyzed by POSTGEN and parasitic capacitive loading due to physical signal layout is then input back to FUNSIM for more precise simulation. The effect of layout and package inductance is not taken into account in this modelling but would be required for a complete simulation, especially when designing chips operating at speeds higher than 15-20 MHz.

4.2. DETAILED DESIGN

4.2.1. System Modules

4.2.1.1. Opcodes

Since this processor is designed as a semi-RISC machine, the instruction set is minimal. Careful analysis of the actual application code (appendix C) and experience with

this type of application determined what instructions are either required or could be easily reduced to several simpler instructions. As this processor is in view of ADC and I/O processing, memory access instructions were found to be particularly useful and this is reflected by the multitude of ways to access memory. Additionally, bit testing is often needed in such applications and a specialized instruction (BTST) for bit comparison is provided with the data register specifying the bit number and one address register containing the address of the byte to be verified. Standard MOVE instructions and boolean instructions (AND, OR, SHIFT, CMP) are provided. Arithmetic operations ADD and SUB are provided as well with negative and zero flags, although the carry flag has been left out. This is in fact due to the design methodology of code analysis, where the code did not need the carry flag functionality and so it was left out. This oversight has little effect on architecture and could easily be integrated as an upgrade. As the original processor is a Motorola MC68000, it was decided to make the opcodes and instruction names very similar to that of the MC68000. This simplifies converting the program to this design's processor, with very few changes required in the source code to execute properly. One of the major factors affecting system designers' choice of processor is indeed portability of application code. This is very apparent in the MC680x0 family for example where each new processor supports almost all previous opcodes.

The source code of the present system, listed in appendix C, which is to be implemented was analyzed and the opcodes in Table 4 were determined to be those absolutely required by the code. Further simulations and analysis could determine further opcodes or alternate opcodes preferable to these for reasons of efficiency. One of the design goals here was to keep similar opcodes to those of the MC68000.

In this design there are sixteen supported opcodes (see table 4). Each original opcode is converted from the MC68000 format to a one of sixteen recoded vector by the "instruction code compression unit". (see Table 3) This simplifies processing within the microcode by pre-grouping and pre-differentiating opcodes. For example, in Table 3 vectors 2 and 3 are differentiated into vectors 7 and F. Though the 4 MSB of their opcodes are the same, the lower bits of the opcode may be found to be different. This differentiates the MOVE.w (Asss), Dddd and MOVE.w #<X>, Dddd instructions for the microcode. Standard branch and jump instructions are supported in two's complement format, with short branches contained within 8 bits encoded within the opcode and long branches contained in the next instruction's 16 bits. As these opcodes compose a two operand language, one of the source registers must also be a destination register.

Table 2 shows the distribution of opcode functionality. Separate addressing modes for the same opcode are counted as separate opcodes in this table. As can be seen, most of the operations are RISC-like. The two exceptions are for specialized software-specific code simplification. The opcode functionality is divided into four quarters, one is for computations, one for access to and from memory, and two for branch control. The latter two are almost equally divided between long and short branches. Such a proportion seems reasonable for a data acquisition system, as it execute do many data movements, some computations, and some program flow control. Further analysis of the program would probably reveal that it is possible to eliminate the long branches. Keeping the RTS and JMP which were considered long branches would help eliminate the other long branches. This would reduce each class to one third of operations and the total number of opcodes by 19%. One of the two register-to-memory (computation) opcodes is the BTST operation, used to verify a bit status such as the ready bit from a data conversion hardware unit. It is specifically useful for such applications.

Opcode Functionality	Quantity	Percentage	RISC-like?
memory-to-register (move)	5	19%	Yes
register-to-register (computation)	7	27%	Yes
branching (long)	7	27%	Yes
branching (short)	5	19%	Yes
register-to-memory (computation)	2	8%	No
Total	26	100%	

Table 2, Opcode Functionality Breakdown

4.2.1.2. MicroProgram Controller

This section describes the microprogram controller MPC unit as seen within Figure 14. The microprogram controller is responsible for sequencing all operations within the chip. Essentially, it may be viewed as a software program, the microprogram, which controls what happens in the chip by providing all control signals and some data signals to other functional blocks in the chip in the time-appropriate order.

Vec ID	4MSB of Opcode		Recoded Vector	Changes Bits #
0	3	→	3	none
1	1	→	1	none
2	3	→	7	2
3	3	→	F	2, 3
4	2	→	2	none
5	0	→	0	none
6	C	→	C	none
7	8	→	8	none
8	D	→	D	none
9	D	→	9	2
A		→	9	none
B	E	→	E	none
C	E	→	E	none
D	B	→	B	none
E	4	→	4	none
F	4	→	5	0
10	6	→	6	none
11	6	→	A	2, 3
12	6	→	6	none
13	6	→	6	none
14	6	→	6	none

Table 3, Vector Number Coding

In this implementation, the MPC is synthesized by the computer program SYNFUL from a state listing into a Sea-of-Gates. This is a group of combinatorial and sequential gates combined in an appropriate fashion to generate the desired sequence of signals. For more dense but slower implementations, the MPC may be synthesized into a ROM or an OR-AND PLA. The current state of the system is given by a "state counter" which is controlled by the internal logic and stepped by an external clock. This is referred to as a "MicroCoded Control Pod".

				MSB			LSB		*
Vec ID	Opcode			FEDC	BA98	7654	3210		
0	MOVE.W	Dsss,	(Addd)	0011	ddd0	1001	0sss		
1	MOVE.B	(Asss),	Dddd	0001	ddd0	0001	0sss		
2	MOVE.W	(Asss),	Dddd	0011	ddd0	0001	0sss		
3	MOVE.W	#<X>, word2	Dddd bbbb	0011 bbbb	ddd0 bbbb	0011 bbbb	1100 bbbb		
4	MOVEA.W	#<X>, word2	Addd bbbb	0010 bbbb	ddd0 bbbb	0111 bbbb	1100 bbbb	1	
5	BTST.B	Dsss,	(Addd)	0000	sss1	0001	0ddd	2	
6	AND.W	Dsss,	Dddd	1100	ddd0	0100	0sss		
7	OR.W	Dsss,	Dddd	1000	ddd0	0100	0sss		
8	ADDA.W	#<X>, word2	Addd bbbb	1101 bbbb	ddd0 bbbb	1111 bbbb	1100 bbbb		
9	ADD.W	Dsss,	Dddd	1101	ddd0	0110	0sss		
A	SUB.W	Dsss,	Dddd	1001	ddd0	0100	0sss		
B	LSL.W	#8,	Dddd	1110	0001	0100	1ddd	3	
C	LSR.W	#8,	Dddd	1110	0000	0100	1ddd	3	
D	CMP.W	Dsss,	Dddd	1011	ddd0	0100	0sss		
E	RTS			0100	1110	0111	0101		
F	JMP	<X> word2		0100 bbbb	1110 bbbb	1111 bbbb	1001 bbbb		

Table 4A, Listing of Processor Opcodes, Part 1 of 2

The state counter may be stepped to the next state or to any state in the microprogram, and this next state may or may not depend on the current value of inputs to the microprogram. Thus, it is a state machine where the system clock and various inputs control the present state, and whose outputs control the rest of the system during a particular state.

This system's microcode is single-level, not multi-level as in some systems that have a nanocodem, for example, where each microcode step is written as several nanocode steps [6]. The microcode has six inputs in addition to the clock and the reset signal. There are 24 output bits in addition to a 16 bit output bus.

Vec ID	Opcode			MSB		LSB		*
				FEDC	BA98	7654	3210	
10	BRA	<x>		0110	0000	bbbb	bbbb	4
11	BSR	<x>		0110	0001	bbbb	bbbb	4
12	BNE	<x>		0110	0110	bbbb	bbbb	4
13	BEQ	<x>		0110	0111	bbbb	bbbb	4
14	BMI	<x>		0110	1011	bbbb	bbbb	4
10	BRA	<X>		0110	0000	0000	0000	
			word2	bbbb	bbbb	bbbb	bbbb	4
11	BSR	<X>		0110	0001	0000	0000	
			word2	bbbb	bbbb	bbbb	bbbb	4
12	BNE	<X>		0110	0110	0000	0000	
			word2	bbbb	bbbb	bbbb	bbbb	4
13	BEQ	<X>		0110	0111	0000	0000	
			word2	bbbb	bbbb	bbbb	bbbb	4
14	BMI	<X>		0110	1011	0000	0000	
			word2	bbbb	bbbb	bbbb	bbbb	4

Table 4B, Listing of Processor Opcodes, Part 2 of 2

*0	b above means a single bit equal to 0 or 1
*1	not sign extended, actual MC68000 would be .L but this is a 16 bit processor
*2	BTST Z-flag = not [bit #Dsss] in (Addd), only on lower 8 bits
*3	this is equivalent to swapping upper and lower bytes
*4	for BRANCH instructions:
	if lower byte is 0: this is a long branch, 2's complement <X> displacement is in next word
	if lower byte not 0: this is a short branch, 2's complement <x> displacement is in opcode low byte

Table 4C, Description of Nomenclature Used in Table 4A and Table 4B

The output bus is actually a constant 15 bits plus one bit which changes according to the microcode. This format has been kept this way because this output bus is used to generate constants for the system and may in the future need more than just one bit to

generate all possible constants. Figure 14 shows the system architecture diagram repeated, with the addition of all signals and shows the flow of signals; Table 5 shows the functionality of each signal.

The microcode was generated by the computer with the option of using LSSD scan methodology, in which all input and output signals of the MPC block are passed through a chain of registers. Then, by single stepping the clock and shifting these signals in series, the actual system state may be traced. Alternately, the system may be placed into any state by shifting in any desired pattern of bits. This is described in the previous section on scan methodologies as a Scan-In-Scan-Out design.

The hardware capability of the DPSRAM which stores the registers is exploited in the microcode by doing simultaneous access of both source operands when possible. This is required in binary operations such as ADD which require two source operands. Thus the DPSRAM sources the two operands onto BUS A and BUS B, which are in turn connected to the ALU.

In order to guarantee proper system operation, the microcode implements ALU operations in three clock cycles. The first cycle fetches the sources for the operation and processes them. The second cycle latches the result into the output register A. The third cycle stores the result back into the DPSRAM registers. The system was actually designed to combine the last two operations into a single clock cycle, but this is straightforward and simply a question of editing the microcode listing. The disadvantage of combining the last two operations is the resulting tighter timing requirements since the data must be available to the DPSRAM in the same cycle that it is latched into the ALU output register.

Although no hardware provision is provided for interrupt processing, this would be easy to implement by adding one or more interrupt signal inputs to the core. Then the microcode would check the interrupt status between instructions and would execute special interrupt microcode if an interrupt was set.

The BTST opcode requires the microcode to transfer data from BUS A to BUS B. This is implemented by using the ALU to add zero to the BUS A data and outputting the result via BUS C onto BUS B. This convoluted procedure adds one to two clock cycles to the processing of the BTST instruction. Simplification and speed enhancement

can be obtained by either changing the bit-encoding sequence of the opcode, or by providing a direct BUS A to BUS B data path, perhaps via BUS C.

Currently, five clock cycles are used to increment the Program Counter PC. Building the PC as a register with its own increment hardware would reduce this to zero clock cycles but would require additional chip area.

Due to software limitations of the utilized microcode-to-circuit synthesis package, outputs of the signals could not be latched. In other words, if a signal was defined as going high in a particular state, the signal did so correctly, but returned to low in the following state unless redefined as high in this following state. This could have required additional latches for all of the 40 output bits of the MPC. The microcode would have set a signal to one to toggle the state of the output bit to its logical inverse and set the signal back to one again to toggle it back to its normal state. Initialization would be done by tying the system reset line to either the set or the clear line of each output latch depending on whether it is desired to initialize it to one or to zero. The signal would also have to be ANDed with the system clock each time to cover the case where the toggle signal is asserted in two consecutive states. This approach complicates design because forgetting to re-toggle a signal back to its normal state once will then invert all further uses of that bit. In an absolute definition of the signal the bit would reset itself to the proper setting at the next reference. It would also be necessary to manually add the latching circuitry for each output bit. It should be noted that a more recent release of the synthesis software corrects this problem by providing latching outputs.

Another approach would be to redefine all bits that need to stay one in their normal state at each state. This would require a fair amount of additional work to implement and would be prone to errors, but this cost could be balanced with the cost of implementing the output latching circuitry mainly with respect to chip real estate. A further refinement would be to add inverters to signals normally high and thus all signals would return automatically to their default state unless defined otherwise. This approach simplifies both the work and does not require much more real estate. In this design, the latter approach was taken, requiring only a minor rewrite of the microcode and the addition of 8 inverters. This approach works quite well since most signals such as gate controls will be active for one to three cycles only and then release the gate. For example, a bus access control might go low for two cycles in order for the data to access the bus, be used by its destination and then return back to its normal disabled state.

Figure 14, System Architecture with Signals

Signal	I/O	Description
VDD	I	power
VSS	I	power
BRANCH	I	1 = Branch code & flags force a branch
NULL	I	1 = 8 LSB of IR = 0, this means branch is long
CODE(3:0)	I	recoded instruction vector
CK	I	clock to sequence to the next state
RST	I	resets state counter to state 0
ILD	O	load Instruction Register IR
IRG	O	enable IR to BUS A link
AGA	O	enable BUS C from ALU to BUS A link
AGB	O	enable BUS C from ALU to BUS B link
HFB(0:15)	O	a number output from the MPC to BUS B, used to supply constants, linked by BBE
ASV	O	loads ALU output register, edge trigger
AT(0:1)	O	specifies ALU operation, see section on ALU
CSD	O	0 = enable device interface circuitry DVI
BBE	O	0 = enable HFB to BUS B link, 1 = hi-Z link
ALX	O	0 = link BUS A to ALU BUS A input, 1 = link HFB to ALU BUS A input
LFB	O	0 = point to Program Counter PC, 1 = point to Stack Pointer SP. 16+LFB is the address accessed in the DPSRAM.
BW	O	0 = word access is being done, 1 = byte access
IR0	O	1 = force ALU adder to add, 0 = give ALU adder control to IR14 from Instruction Register IR
BAA	O	0 = select data reg, 1 = select addr. reg Port A
BBA	O	0 = select data reg, 1 = select addr. reg Port B
PAW	O	start Port A write cycle on rising edge
PBW	O	start Port B write cycle on rising edge
RW	O	0 = write cycle, 1 = read cycle
RAS	O	0 = select BSB, 1 = select BAA&DST as addr.
RBS	O	0 = select HFB, 1 = select BBA&SRC as addr.

Table 5A, Description of Signals To and From the MPC Unit, Part 1 of 2

Signal	I/O	Description
AEN	O	0 = link DPSRAM Port A data out to BUS A, 1 = hi-Z
BEN	O	0 = link DPSRAM Port B data out to BUS B, 1 = hi-Z
DS	O	data strobe, 0 = asserted as below, 1 = invalid if RW is 0 = data bus output valid on write, if RW is 1 = data bus expected valid for read
STM(9:1)	O	current state # of state counter
SCAO	O	testpins scan output
SCAI	I	testpins scan input
SCAM	I	testpins scan mode, 1 = test, 0 = normal mode

Table 5B, Description of Signals To and From the MPC Unit, Part 2 of 2

Signal	I/O	Description
BUS A(15:0)	I	from BUS A, used as external data bus
NF	I	Negative-flag
NFB	I	Negative-flag barred
ZF	I	Zero-flag
ZFB	I	Zero-flag barred
ILD	I	Load Instruction Register command
IR(15:0)	O	Instruction Register's bits
SRC(2:0)	O	source register # (be it data or address)
DST(2:0)	O	destination register #(be it data or address)
BRANCH	O	1 = should branch, 0 = do not branch, given the present Bcc & flags
NULL	O	1 = lower 8 bits of IR = 0 = long branch for Bcc
CODE(3:0)	O	recoded instruction vector, specifies instruction to execute for microcode

Table 6, Description of Signals To and From the BRA Unit

4.2.1.3. Branch Control Unit

This section describes the Branch Control Unit (BRA) and the Instruction Register (IR), as seen within Figure 14. The purpose of the BRA block is threefold. First, the BRA

block contains a register **IR** which holds the opcode presently being executed. A register is necessary because the opcode is used by the hardware over several clock cycles. For example, source and destination register addresses are sourced from the instruction and this is required for several cycles. Second the **BRA** unit contains the hardware to translate the opcodes into the internal recoded vector number, as explained in the previous section "Opcodes". This does preliminary opcode decoding to simplify the microcode rather than forcing the microprogram to decode the opcode. As well, the opcodes are reduced to a four bit opcode thus reducing microcode hardware logic. For example, the **ADD** and **SUBtract** operations are grouped together with a bit sent directly to the **ALU**'s adder indicating whether to add or subtract the numbers. The opcode preprocessing by the **BRA** unit of the add/subtract opcode reduces the microcode processing to place the two numbers to be added to the **ALU** and to retrieve the resulting number from the **ALU**. Finally, the **BRA** unit performs pre-processing for the microcode by comparing "branch" opcodes to the **ALU**'s output condition flags and providing the microcode with a bit indicating whether or not to branch. Thus the microcode does not know if the branch was, for example, on the zero flag but only whether or not it was successful. The **BRA** also analyzes the lower 8 bits of the opcode and determines whether they are all zero, which for a branch instruction signifies a long branch is required. This short or long indicator bit is provided directly to the microcode, saving it from having to use the **ALU** to check these bits. Thus the microcode does not know all the details of operation and instruction decoding is shared between hardware and microcode.

This behavioural model was simulated. The next step was to manually generate a circuit to emulate the required specifications. Analysis of the behavioural language code led to direct translation into hardware. Table 7A and Table 7B summarize these hardware functions. The code translation circuit was developed by manually noticing trends in the bit-translation maps between opcodes and recoded vectors and manually synthesizing the functionality into a circuit. This could also have been synthesized by one of several programs commonly available such as ViewLogic or Tango-PLD.

4.2.1.4. Memory Facility

This section describes the on-chip memory facility composed of the **RIM** and **RIF** units, as seen within Figure 14. The **RIM** unit is a **DPSRAM** memory allowing two simultaneous independent reads or writes, except for simultaneous read and write to or from the same memory location.

Function	Equation
FUNCTION 1	Determine if Branch should be taken
Invert Signals	$IRxB = NOT(IRx)$
	where $x = 15, 13, 12, 11, 10, 9, 8$
BRANCH should occur	$BRANCH = A \mid B$
RTS & JMP: Opcode(15:12) = 4	$A = IR15B \& IR14 \& IR13B \& IR12B$
Branch code & set	$B = C \& D$
All Branches must have Opcode(15:12) = 6	$C = IR15B \& IR14 \& IR13 \& IR12B$
Branch code set	$D = D1 \mid G \mid D2$
	$D1 = E \mid F$
	$D2 = H \mid I$
BRA: Opcode(11:8) = 0	$E = IR11B \& IR10B \& IR9B \& IR8B$
BSR: Opcode(11:8) = 1	$F = IR11B \& IR10B \& IR9B \& IR8$
BNE: Opcode(11:8) = 6 with Z-Flag reset	$G = G1 \& ZFB$
	$G1 = (IR11B \& IR10 \& IR9 \& IR8B)$
BEQ: Opcode(11:8) = 7 with Z-Flag set	$H = H1 \& ZF$
	$H1 = (IR11B \& IR10 \& IR9 \& IR8)$
BMI: Opcode(11:8) = 11 with N-Flag reset	$I = I1 \& NFB$
	$I1 = (IR11 \& IR10B \& IR9 \& IR8)$
FUNCTION 2	Determine if is short or long branch
	$NULL = NOP(N1, N2)$
	$N1 = BUS A(7) \mid BUS A(6) \mid$ $BUS A(5) \mid BUS A(4)$
	$N2 = BUS A(3) \mid BUS A(2) \mid$ $BUS A(1) \mid BUS A(0)$
FUNCTION 3	Provide Instruction Register IR

Table 7A, Hardware Implementation of the BRA Unit, Part 1 of 2

Function	Equation
FUNCTION 3 (cont'd)	Use 16 DP D-Flip-Flops wired as: $\text{Data_In}(i) = \text{BUS } A(i), i = 0..15$ $\text{Data_Out}(i) = \text{IR}(i), i = 0..15$ $\text{Data_Out_Bar}(i) = \text{IRB}(i), i = 0..15$ $\text{Clock}(i) = \text{ILD}$
	Note that IR bits may be referred to as IR(i) or IRi. Ranges are IR(i:j).
FUNCTION 4	Output the source and destination number contained in the opcode
Source Register	$\text{SRC}(2:0) = \text{IR}(2:0)$
Destination Register	$\text{DST}(2:0) = \text{IR}(11:9)$
FUNCTION 5	Recode the Opcode for the MPC
Vector bit 0 (LSB)	$\text{CODE}(0) = \text{IR}0 \mid$ $(\text{IR}12\text{B} \& \text{IR}7 \& \text{IR}13\text{B})$
Vector bit 1	$\text{CODE}(1) = \text{IR}13$
Vector bit 2	$\text{CODE}(2) = (\text{IR}14 \& \text{NOT}(\text{J}) \& \text{NOT}(\text{K})) \mid \text{L}$
Vector bit 3 (MSB)	$\text{CODE}(3) = \text{IR}15 \mid \text{J} \mid \text{M}$
	$\text{J} = \text{IR}15\text{B} \& \text{IR}14 \& \text{IR}9\text{B} \& \text{IR}8$
	$\text{K} = \text{IR}14 \& \text{IR}12 \& \text{IR}7\text{B}$
	$\text{L} = \text{IR}13 \& \text{IR}12 \& \text{IR}4$
	$\text{M} = \text{IR}13 \& \text{IR}12 \& \text{IR}5 \& \text{IR}4$
Gate Count Summary:	SUDS designation : Quantity
	AND2: 4
	AND3: 4
	AND4: 9
	OR2 : 5
	OR3 : 2
	OR4 : 2
	NOR2: 1
	INV : 9
	MX2 : 16

Table 7B, Hardware Implementation of the BRA Unit, Part 2 of 2

Address	Functional Usage
0-7	D0-D7 data registers
8-15	A0-A7 address registers
16	Program Counter PC
17	Stack Pointer SP
18-23	Stack Registers (programmable)
24-192	DPSRAM for application programs

Table 8, On-chip DPSRAM Memory Map

The memory size is 192 words of 16 bits each and thus eight address bits are required to access all locations. The main purpose of the memory is to provide emulation of the 8 data and 8 address registers within the MC68000. Additionally, the program counter (PC), stack pointer (SP), and stack are located within the memory. The memory map for the DPSRAM usage is shown in Table 8. The internal registers and memory could all be implemented using registers and external memory, but these latter are very area and time consuming as compared to DPSRAM memory. The drawback of using DPSRAM memory used is that it takes about 50 nanoseconds to access or store data, whether for a read or a write, as shown in Figure 17 and Table 12. Because Dual Port RAM is used, two locations may be accessed simultaneously. This is very important as many instructions use two registers as sources. Note that, as two locations may be accessed simultaneously, one could conceivably perform a write of a previous instruction result with a read for a current instruction, thus allowing some parallelism. Another function of this DPSRAM is to provide programs with 174 on-chip memory locations for application program usage. Access is provided by reserving a memory address range for the internal DPSRAM (see description of the DVI). These locations could be used as a cache or for register windowing, which would be facilitated with minor additional window offset hardware. Unless some hardware upgrade is implemented or the operating system protects the first 18 locations in memory, it is possible to overwrite the registers or PC or SP by writing to these locations in the DPSRAM. This could also be used as a loophole by programmers.

The RIM model for the DPSRAM was generated automatically by the CLASSIC program described in Chapter 4.1 and includes precise timing checks for simulation. The timing is shown in Figure 17 and Table 12.

The **RIF** model generates the necessary timing using combinatorial logic to simplify microcode access to the **DPSRAM**. Thus, the microcode simply requests a read or write but does not have to manage detailed timing. The **RIF** block also generates all static signals for operating the **DPSRAM**, namely programming it to the desired access mode. As well, two multiplexers are provided to select the source for the **DPSRAM** address, a process that is under microcode control. This is easily seen in Figure 14. The **Port A** address is taken from **BUS B** or from the destination register contained within the opcode in the **IR** and bit **BAA** from the microcode indicating whether a data (**BAA** = 0) or address register (**BAA** = 1) is being accessed. Similarly, the **Port B** address is taken from the **LFB** microcode signal plus an offset of 16 (**LFB** = 0 means **PC**, 1 means **SP**) or from the source register contained within the opcode in the **IR** and bit **BBA** from the microcode indicating whether a data (**BBA** = 0) or address register (**BBA** = 1) is being accessed.

Timing and circuitry are shown for the **RIF** in Figure 15 and Figure 16 and are described in the next paragraph. This circuitry was generated manually by observing the behaviour of the model first used to simulate the system and by observing the **DPSRAM**'s timing specifications as given in Figure 17. The circuitry is straightforward with one exception. To generate the proper timing, an external 25MHz clock (**EXTCLK**) is broken into cycles of 3 half-cycles, yielding a clock **INTCLK** with a period 1.5 times that of the external clock. This is needed in order to generate the proper memory access cycles such as pre-charge. This **INTCLK** is then viewed by the chip as the system clock with a frequency of 16.7MHz. The clock is then further processed by the bus interface module **DVI** in order to add wait states for external devices interfacing, at which point it is called **MPCCLK**. This clock signal **MPCCLK** clocks the microcontroller's state machine with **DPSRAM** access and external device access transparent to the microcode. The system's design could perhaps be optimized to take advantage of the asymmetry in the **INTCLK**.

The **DPSRAM**'s modes of operation, timing diagrams, and timing specifications are given in Table 11, Figure 17, and Table 12. The waveforms synthesized by the **RIF** circuitry to meet these specifications are shown in the top half of Figure 15. The **DPSRAM** has two access paths, namely **Port A** and **Port B**. As seen on Figure 15 bottom half, the **RIF** clock generating circuitry for **Port A** is composed of two D-type flip-flops and three **AND** gates. Together, they generate the **INTCLK**, **PAWi**, **PARE**, **PAP2** signals necessary to interface to the **DPSRAM** for the **DPSRAM**'s **Port A**. Required inputs are **EXTCLK**, the externally provided 25 MHz clock, **RESET** to clear the flip-flops to a known state, **RW** to choose read or write timing waveforms, and **CSRAM** to trigger when the **DPSRAM** has been selected for interface.

RIF Timing Diagram

RIF Circuitry

Figure 15, RIF Model Timing and Circuitry P1 of 2

2 of/de 2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

PAXS = PAYS = PAP1 = 1

RIF Address Bus Multiplexer

The previous circuits were for PORT A
For PORT B, copy the circuitry but:

Replace	By
PAWi	PBWi
PAP2	PBP2
PARE	PBRE
BSB(7:0)	HFB(7:0)
BAA	BBA
DST(2)	SRC(2)
DST(1)	SRC(1)
DST(0)	SRC(0)
PAAD(7:0)	PBAD(7:0)
PAXS	PBXS
PAYS	PBYS
PAP1	PBP1

Additionally it is necessary to:

1. invert CSRAM to get CSRAM
2. invert RW to get RW

RIF Additional Circuitry

Figure 16, RIF Model Circuitry P2 of 2

The input address source multiplexers and the constant values of **PAXS**, **PAYS**, **PAP1** required to drive the DPSRAM are shown at the top of Figure 15. The multiplexers are 8 one-bit multiplexers in parallel whose outputs onto the **PAAD** bus are selected by the **RAS** signal. For **Port B**, the same circuit is used as for **Port A**, but with the **Port A** signals substituted for the **Port B** signals as shown in the table in the bottom half of Figure 16. All these signals are required for proper microcode access to the DPSRAM by the **MPC** unit, or, stated differently to simply assert the **RW** and **RAS** or **RBS** signals.

Signal	I/O	Description
VDD	I	power supply
VSS	I	power supply
PADI(0:15)	I	Port A data input
PBDI(0:15)	I	Port B data input
PAW(0:15)	I	Port A bit write enable mask
PBW(0:15)	I	Port B bit write enable mask
PAXS	I	from RIF, Port A mode (see text)
PBXS	I	from RIF, Port B mode
PAYS	I	from RIF, Port A mode
PBYS	I	from RIF, Port B mode
PARE	I	0 = enable Port A output, 0 = hi-Z
PBRE	I	0 = enable Port B output, 0 = hi-Z
PAP1	I	from RIF, Port A mode
PAP2	I	from RIF, Port A mode
PAAD(0:7)	I	from RIF, Port A address
PBAD(0:7)	I	from RIF, Port B address
PBP1	I	from RIF, Port B mode
PBP2	I	from RIF, Port B mode
PAOT(0:15)	O	tri-state data from Port A
PBOT(0:15)	O	tri-state data from Port B

Table 9, Description of Signals To and From the RIM Unit.

Signal	I/O	Description
DST(2:0)	I	selects destination register Port A
BAA	I	0 = select data reg, 1 = select addr. reg Port A
RAS	I	0 = select BSB , 1 = select BAA&DST as addr.
SRC(2:0)	I	selects destination register Port B
BBA	I	0 = select data reg, 1 = select addr. reg Port B
RBS	I	0 = select HFB , 1 = select BBA&SRC as addr.
BSB(7:0)	I	Port A address if selected by RAS
HFB(7:0)	I	Port B address if selected by RBS
PAW	I	start Port A write cycle on rising edge
PBW	I	start Port B write cycle on rising edge
PAAD(0:7)	O	to RIM , Port A address
PBAD(7:0)	O	to RIM , Port B address
PAWW(0:15)	O	to RIM , bit write mask for Port A
PBWW(0:15)	O	to RIM , bit write mask for Port B
PAXS	O	to RIM , Port A mode
PBXS	O	to RIM , Port B mode
PAYS	O	to RIM , Port A mode
PBYS	O	to RIM , Port B mode
PAP1	O	to RIM , Port A mode
PAP2	O	to RIM , Port B mode
PBP1	O	to RIM , Port A mode
PBP2	O	to RIM , Port B mode

Table 10, Description of Signals To and From the **RIF** Unit.

The analysis of the different operating modes of memory (see Table 11) shows a pre-charge mode which must be performed before bus reads. It is used to "pre-charge" the lines of the **DPSRAM** so that a faster read may be performed. Figure 17 shows the **DPSRAM**'s read and write timing cycle which is fairly standard, with the exact timing estimates given in the **DPSRAM**'s model listing, appendix A. The model listing also shows the following estimates for the **DPSRAM**. The area estimate is given as 110*120 mils and there are 31138 transistors. Input and output loading values are given in picofahads and power usage calculations show 570mW DC plus 32mW AC at 10MHz.

Figure 17, DPSRAM Read and Write Cycle Timing

 * TIMING SPECIFICATION OF CMOS4 DUAL PORT SRAM *
 * (Worst Case) *

PORT A WRITE CYCLE

Symbol	Description	Value
tAVAVW	Address Valid to Address Valid Write Cycle (min):	41.23
tAVWH	Address Valid to Write Enable High (min):	17.74
tAXQX	Address Invalid to Output Invalid (min):	0.00
tWLAX	Write Enable Low to Address Invalid (min):	12.43
tWHWL	Write Enable High to Write Enable Low (min):	11.07
tDVWL	Data Valid to Write Enable Low (min):	11.07
tWLDX	Write Enable Low to Data Invalid (min):	4.68
tRHDI	Read Enable High to Output = DATA IN (max):	6.00

PORT B WRITE CYCLE

Symbol	Description	Value
tAVAVW	Address Valid to Address Valid Write Cycle (min):	40.78
tAVWH	Address Valid to Write Enable High (min):	17.74
tAXQX	Address Invalid to Output Invalid (min):	0.00
tWLAX	Write Enable Low to Address Invalid (min):	12.65
tWHWL	Write Enable High to Write Enable Low (min):	10.39
tDVWL	Data Valid to Write Enable Low (min):	10.39
tWLDX	Write Enable Low to Data Invalid (min):	4.68
tRHDI	Read Enable High to Output = DATA IN (max):	6.00

PORT A READ CYCLE

Symbol	Description	Value
tAVAVR	Address Valid to Address Valid Read Cycle (min):	46.59
tRHRL	Read Enable High to Read Enable Low (min):	28.76
tAVRL	Address Valid to Read Enable Low (min):	38.56
tRHQV	Read Enable High to Output Valid (max):	28.76
tRHQX	Read Enable High to Output Invalid (min):	0.00
tRLAX	Read Enable Low to Address Invalid (min):	8.04
tAVQV	Address Valid to Output Valid (max):	38.56
tAXQX	Address Invalid to Output Invalid (min):	0.00

PORT B READ CYCLE

Symbol	Description	Value
tAVAVR	Address Valid to Address Valid Read Cycle (min):	48.72
tRHRL	Read Enable High to Read Enable Low (min):	30.88
tAVRL	Address Valid to Read Enable Low (min):	40.68
tRHQV	Read Enable High to Output Valid (max):	30.88
tRHQX	Read Enable High to Output Invalid (min):	0.00
tRLAX	Read Enable Low to Address Invalid (min):	8.04
tAVQV	Address Valid to Output Valid (max):	40.68
tAXQX	Address Invalid to Output Invalid (min):	0.00

Table 11, DPSRAM Timing Specifications for Port A

PAXS PAYS (PBXS) (PBYS)	PAWi (PBWi)	PARE (PBRE)	PAP2 (PBP2)	Mode (shaded unused)
0	0	0	0	StandBy/ PreCharge
1	0	0	0	PreCharge
1	1	0	1	Write
1	0	1	0	Read
1	1	1	0	Write/ Read

Table 12, DPSRAM Operation Modes

4.2.1.5. Arithmetic Logic Unit

This section describes the Arithmetic and Logic Unit ALU implemented as two blocks, namely the AIF and ALU blocks, as seen in Figure 14. The purpose of this block is to provide a facility for the mathematical manipulation of numbers. Although fairly simple in structure, it can perform all necessary computations for a typical application program of the scope intended for this system. The ALU can perform two basic classes of operation, boolean and addition/subtraction. As can be seen in the ALU detail diagram in Figure 18 and Figure 19, all possible operations are always performed in a single clock cycle and the appropriate operation is selected by an output multiplexer.

This input multiplexer is controlled by the AT bus from the microcode, with details of encoding given in the table. Then, in a second stage, the result is loaded into an output latch, register A, and status flags are set to reflect the condition of the result. Two such flags are implemented here, namely the Z or zero flag and the N or negative flag. Other flags, such as carry, have been omitted as explained in the Section 4.2.1.1, but would be easy to implement. The Z-flag indicates all bits of the output data are zero if it is one. If a bit test (BTST) is being performed, the Z-flag will be one if the indicated bit is zero. The N-flag indicates bit 15 of the output data is one, which may be interpreted as the number having a negative value in two's complement mathematics.

The output register allows for the common case where two registers are the sources for the numbers to process by the ALU and a register is also the destination. At least one operand has to be released by the internal DPSRAM and the ALU's output would change before the result could be stored back into the DPSRAM, if there was no output register. Thus, in the first clock cycle the two source operands are presented to the ALU and processed. The second clock cycle stores this result in the output **Register A**, and the third clock cycle stores the result into a destination register in the DPSRAM. In practice, this timing depends on the microcode and could be compressed into two cycles by combining the second and third cycles into a single cycle as discussed in the microcode controller section. As the assembly language is a two operand language, the destination register will be the same as one of the source registers.

It is to be noted that all ALU operations take a single cycle to execute internally to the ALU, although at least two are required to store the result. This makes it easy to calculate timing latency for interrupt processing, for example. The register latch control **ASV** also latches in the resulting condition flags. The **AIF** is an add-on block to the ALU and in fact is a subsection of it. It provides the ALU with high-impedance interface from **BUS C** to both **BUS A** and **BUS B**, independently. It adds the input multiplexer onto the **BUS A** input of the ALU, allowing selection of either system **BUS A** or **HFB** bus from the microcode **MPC**. This allows the microcode to specify constant numbers to the ALU, such as 1 when incrementing the program counter, for example.

Finally, the ALU ORs **IR14** and **IR0** to create new signal **NIR14** to the ALU. **NIR14** selects whether the adder will add or subtract the two operands. **IR14** is bit 14 of the instruction register and is used to differentiate whether the opcode processed is add or subtract. **IR0** is a bit from the microcode allowing it to override the ALU add/subtract feature to add. This is used, for example, when incrementing the program counter under microcode control and the currently loaded instruction in the instruction register could be a subtract. To do a subtraction, the microcode generates a 2's complement number and does an addition.

The following section describes the ALU circuit as shown in Figure 18 and Figure 19. Figure 18 shows the first stage of the ALU. There are five blocks composing this first stage and they perform the five operations the ALU supports. The inputs to the ALU are presented on buses **BUS A** and **BUS B**. The ALU's input **BUS A** is actually controlled by the **AIF** block, which multiplexes system **BUS A** and the **HFB** bus from the **MPC** as

chosen by the **ALX** signal. The **ALU** supports all 16 bits provided on **BUS A** and **BUS B** and its output is 16 bits wide. All five possible functions are always computed and are forwarded to the multiplexer shown on the right of Figure 18. The output of this multiplexer is selected by the **AT0** and **AT1** signals as set by the **MPC** unit. The first function is the boolean "AND" operation and is provided by AND gates. The second function is the boolean "OR" operation and is provided by OR gates. The third is to "SWAP" the upper and lower bytes of **BUS B**. This is achieved by wiring the bits in their "swapped" order directly to the output multiplexer.

The fourth function is a 16 bit adder and subtracter. The add capability is provided by placing 16 full-adder blocks in parallel and letting the carry ripple through the chain. Recall that as described in the memory facility section, the system clock is limited to a 60 nanosecond cycle by the **DPSRAM**. Thus it is acceptable to use a carry ripple mechanism in the adder since the total ripple time is less than 60 nanoseconds. The carry ripple is actually one of the highlights while performing chip simulation. To provide add or subtract capability to the **ALU**, the carry input to the lowest bit is controlled by a signal called **IR14**. In addition, the **IR14** bit controls a chain of multiplexers and will select **BUS A** negated as input to the full-adders. Together, these two processes compose a 2's complement computation, since the inversion of **BUS A** is a 1's complement and the input carry bit adds 1 to the 1's complement. Thus, if **IR14** is set to 0, then the operation performed is **BUS B - BUS A**; otherwise the operation is **BUS B + BUS A**. Note that the "compare" operation may be performed by doing a subtract operation, setting the condition flags, but not storing the result of the subtraction.

The fifth function provided by the **ALU** is a bit test, or **BTST**. This function uses an eight bit multiplexer and an inverter. The bit number to test is specified on **BUS A** and the data to test provided on **BUS B**. The bit is selected by the multiplexer and negated by the inverter. Thus if the selected bit is low, then the **BTST** output **BTST_OUT** will be high, which, as described in following paragraphs, sets the **Z**-flag high, indicating the value was zero. This functionality is useful in instrument control to poll a bit's state.

Figure 18 , ALU Detail P1 of 2

Figure 19, ALU Detail P2 of 2

Signal	I/O	Description
CODE(3:0)	I	recoded opcode vector from Instruction Reg
AT(1:0)	I	operation code from MPC as: 0 = $BUS\ B \pm BUS\ A$ (depending on NIR14) 1 = SWAP $BUS\ B$ upper & lower bytes 2 = $BUS\ B\ OR\ BUS\ A$ 3 = $BUS\ B\ AND\ BUS\ A$
BUS A(15:0)	I	processor $BUS\ A$ (via the AIF)
BUS B(15:0)	I	processor $BUS\ B$
ASV	I	load output register and flags on rising edge
NIR14	I	0 = subtract when $AT = 0$, 1 = add when $AT = 0$, it is $IR0\ OR'd$ with $IR14$
IR0	I	0 = $IR14$ decides on add/subtract, 1 = override to add, from MPC
IR14	I	bit 14 of Instruction Register, is bit of opcode that differentiates add and subtract
BUS C(15:0)	O	output data bus
NF	O	Negative-flag, = bit 15 of $BUS\ C$
NFB	O	Negative-flag barred
ZF	O	Zero-flag, is 0 if all bits of $BUS\ C$ are 0
ZFB	O	Zero-flag barred

Table 13, Description of Signals To and From the ALU Unit.

Figure 19 shows the first stage of the ALU. Three blocks compose this second stage. The first block is as follows. The ALU's output as selected in the first stage is latched into Register A by signal ASV under the MPC's microcode control. Having the output latched allows the system to fetch the source operands in one clock cycle, release them and store the result in a second clock cycle. This is necessary since the source registers are implemented in DPSRAM, which has only two access ports. The output of the ALU via Register A is sent to either $BUS\ A$ or to $BUS\ B$ by asserting either AGA or AGB, high impedance gate controls.

The second block is composed of a NOR gate function, an AND gate, a multiplexer, and a D flip-flop. Together these form the Z- or zero-flag which goes high to indicate that an operation's result is zero. The multiplexer selects the input to the flip-flop as

either the **BTST_OUT** signal from the ALU stage one or as the logical NOR of the lower 8 bits of the ALU's output. This is because the Z-flag is dependant on the output of the ALU except if a BTST operation is being performed. The AND gate examines the opcode currently being processed, and thus controls the input multiplexer to the Z-flag.

The third block is the N-flag which is composed of a single D flip-flop. The input to this flip-flop is bit 15 of the ALU's output. Thus, if the output is negative as interpreted using 2's complement encoding, then the N-flag will be set. Both the N-flag flip-flop and the Z-flag flip-flop are clocked by the ASV signal which also loads Register A.

Signal	I/O	Description
ALX	I	0 = select BUS A to ALU BUS A input, 1 = select HFB from MPC to ALU BUS A input
BUS A(15:0)	I	to ALU BUS A input via BUSCOM as per ALX
HFB(15:0)	I	to ALU BUS A input via BUSCOM as per ALX
BUS COM(15:0)	O	selected bus to ALU BUS A as per ALX
AGA	I	1 = hi-Z BUS C to BUS A, 0 = link BUS C to BUS A
AGB	I	1 = hi-Z BUS C to BUS B, 0 = link BUS C to BUS B
BUS C(15:0)	I	ALU latched output result
BUS A(15:0)	O	output to BUS A as selected by AGA
BUS B(15:0)	O	output to BUS B as selected by AGB

Table 14, Description of Signals To and From the AIF Unit.

4.2.1.6. Bus Interface Controller

This section describes the Bus Interface Controller, also known as the Device Virtual Interface DVI and as the System Bus Controller, as seen in Figure 14. The DVI unit incorporates a flexible architecture which may be used in several fashions. The main purposes of this unit are to arbitrate and drive the external I/O for the chip and to synchronize internal buses. Even though the implementation of the DVI in this project is minimal, the bus interface is a useful block which may be easily upgraded. The bus interface provided is asynchronous and address and data buses are non-multiplexed. By including on-chip address decoding and handshaking functionality, the DVI eliminates most external glue-logic.

The original DVI design incorporated special circuitry to convert data from an 8-bit EPROM into a 16-bit bus format. This was provided because the chip's design was based on the original BarTester's design which dates from 1988 when the commonly available EPROM was 8 bits wide. The MC68000 controller, however, required a 16-bit boot EPROM. To minimize chip count and to simplify EPROM programming it was decided to use a single EPROM with circuitry. This circuitry made the EPROM appear as a 16-bit EPROM to the MC68000. This was implemented instead of using two 8-bit EPROMs in parallel to provide a 16-bit bus. At a later stage in the chip design, the 8 to 16 bit conversion circuitry was dropped as it became apparent that 16 bit EPROMs were commonly available.

There are two main internal components to the DVI. The DVI and its timing are shown in Figure 20 and Figure 21. The first component is a facility which implements a standard handshaking algorithm for chip select (CSx) and data transfer acknowledge (DTACK) as in the MC68000. The second component is an address bus latching circuit explained further on the following page. Please refer to the top of Figure 20 for a timing diagram of the handshaking algorithm. The feature of having on-chip select decoding reduces external glue logic and has in fact been adopted as a standard by Motorola in its new chips such as the MC68332 with the external bus interface (EBI). Motorola has also extended this concept to incorporate on-chip DTACK generation and programmable address mapping. The principle of operation of this circuit is simple yet powerful. By controlling the clock which drives the microcode's state machine, (MPCCLK), rather than driving the state machine directly from the external clock, (EXTCLK), the DVI can place the processor core on hold as long as desired. This is referred to as "inserting wait states", a feature used to pause the processor while an external handshake is being performed. Thus handshaking to slow devices is possible and is triggered by the microcode asserting the Chip Select Drive (CSD) signal. Upon return of control to the microcode it must release the CSD signal or another CSD handshake cycle will occur. This capability could also be used to give another on-chip unit control over the system. The price to pay is that the processor does no work while on hold. This could be refined by utilizing internal parallel processing concepts. As all external I/O fetches go through the DVI, it is the ideal place to implement either a read/write buffer or even a cache or memory management unit.

Figure 20. Timing of and Address Bus Latching in DVI, P1 of 2

Figure 21 , Circuitry of DVI, P2 of 2

Of the currently 8 decoded addresses, the lowest range of \$0000 to \$1FFF (hexadecimal) is mapped onto the internal DPSRAM (CS0) to allow programs to use all of the 192 internal DPSRAM locations. CS4 is mapped as the booting EPROM chip select. The other six address ranges are available for either external access or for address selection of other internal units. Several internal units are of interest in meeting this design's extended goal: an IEEE-488 GPIB interface, a parallel interface, a timer, a custom camera interface, and a microsequencer and glue logic for the analog circuitry. Interestingly, the microsequencer could be the Time Processing Unit (TPU) provided in the MC68332, for example, although this would be an overdesign. The TPU unit is a RISC-based coprocessor designed to alleviate the core processor from real-time control of digital I/O lines and is implemented on-chip in the MC68332. For this project a simple synchronous state machine would be sufficient.

Another feature implemented in the DVI is the gating with latches of the address bus, known as **BUS B** internally and **BUS E** or "address bus" externally. The circuit is given in Figure 20, bottom half. The circuit places the external address lines into high-impedance when no access is occurring and asserts the external address lines when in use. This capability could be used to implement multi-processor bus sharing with some upgrades to the DVI. It is necessary to latch the output address lines because the internal **BUS B** cannot hold the address for external devices. For example, during a read from memory operation it must in one cycle output the address to the external device and in the next cycle, after **DTACK** has been asserted, point to the proper DPSRAM address to fetch the data which will be presented to it on **BUS A**, the external address bus.

It would also be possible to implement a multiplexer between **BUS A** and the external data bus in order to simplify access or storage of bytes, specified by the **BW** signal. For example, upon output of a byte it may be desirable to output the upper or lower byte, depending on whether the address is even or odd, to both the upper and lower bytes of the output data lines in a write cycle. In a read cycle, it may be desirable to select either the upper or lower byte into the lower byte and replace the upper byte with zeros. This functionality is straightforward and would utilize little chip area and was thus not implemented in this project.

Signal	I/O	Description
BUS B(15:0)	I	system BUS B
EXTCLK	I	clock supplied to DVI for driving MPC
RST	I	reset command to chip
CSD	I	1 = disable chip select decoding, 0 = initiate address decoding with handshaking cycle
DTACK0.. DTACK7	I	Data Transfer Acknowledge, channel 0 to 7. Handshake back to processor for data valid.
DTACKDRM	I	DTACK0 signal from DPSRAM
DTACKPEM	I	DTACK4 signal from EPROM
CS0..CS7	O	Chip Select, channel 0 to 7. Matched to DTACK channels. For device selection. Channel 0 reserved for internal DPSRAM. Channel 4 currently connected to EPROM.
CSDRM	O	CS0 signal to DPSRAM
CSEPM	O	CS4 signal to EPROM
BUSE(15:0)	O	Latched address BUS B, latched by DS
DS	O	Data Strobe
MPCCLK	O	Clock for MPC

Table 15, Description of Signals To and From the DVI Unit.

4.2.1.7. EPROM and Clock units

This section describes both the reset and clock generator CKC and the EPROM simulation model named EPROM. These are not shown on the System Architecture diagram Figure 14, but are connected to the "External Data Bus", "External Address Bus", and control signals shown at the bottom of the figure. In order to run and observe functional simulations of the entire system it was necessary to provide a complete system description. Thus it was necessary to provide source opcodes (a program) to be executed by the system, namely an EPROM model, and a clock to sequence the system, namely a square wave generator. This is done in two units which are not meant to be part of the chip but rather part of the system the chip would fit into. Firstly, the CKC unit generates a 10MHz clock (EXTCLK) with a power up reset signal RST. This is implemented with a simple

asynchronous model and is straightforward once in place. The **EPROM** unit actually models both a 16-bit wide EPROM with a 130 nanosecond access time and a **DTACK** handshake circuitry to provide the handshake **DTACKEPM** back to the **DVI**. This is also modeled as an asynchronous model. The contents of the EPROM are both the power-up vectors for when the microcode fetches Program Counter and Stack Pointer values upon power-up, as well as a simple program to jog many of the chip's internal functions.

Signal	I/O	Description
ADD(15:0)	I	address of location to be accessed
EN	I	0 = output enable, 1 = hi-Z output
DTA(15:0)	O	output data from the accessed address, state controlled by EN
DTACKEPM	O	simulated DTACK signal from EPROM
P	O	internal signal to EPROM

Table 16, Description of Signals To and From the **EPROM** Unit.

Signal	I/O	Description
RES	O	reset signal for chip
CLA	O	clock signal for chip
EDI,CLI,RIS	O	internal signals to CKC

Table 17, Description of Signals To and From the **CKC** Unit.

4.3. SIMULATION AND LAYOUT OF DESIGN

4.3.1. Introduction

Simulation of the design was performed after chip architecture definition. The first goal of simulation is to confirm the system architecture validity. Firstly each block is defined, including timing and pinout specifications. Thus, a functional or actual component must be created for a block. Then a description of stimuli, test vectors, and expected response with timing for each signal must be generated. Simulation of the blocks is repeated until they pass all timing and response verifications. System debugging can begin at this stage, preferably also with a set of test vectors and associated timing and responses. Finally, any functional blocks are then replaced by actual circuit implementations and re-verified using the same test vectors and their associated timing and responses. Simulation

such as this defines all necessary signals, and includes precise timing generation and checks. In this design, however, an architecture was designed, discussed, and optimized without computer simulations. Then, the individual blocks were designed and tested out individually using a template of expected responses for each. Once the blocks met specifications, they were connected together and simulated as a whole with manual tracing of signals to ensure proper operation. This is a top-down approach. In actual implementation, some functional blocks were broken into several smaller blocks to compensate for limitations in the simulator software. The following sections describe simulation of each of the system's components followed by the simulation of the entire chip. Although exhaustive simulation of the chip as a system has not been performed, each individual module has been characterized and it is felt more simulations would neither add to the architectural understanding presented nor affect any conclusions of this thesis. The actual timing traces are not included as they are both very long and straightforward. Description of the blocks assumes the reader is already familiar with the system architecture as described in earlier chapters. Chip layout issues are then discussed and an estimate of required chip area performed.

4.3.2. System Units

4.3.2.1. Branch Control Unit

The branch control unit **BRA** was implemented in the synthesis language **FML2** as a behavioural model. Test vectors were included to test each section of the **BRA** unit. Each opcode and the status of the **N** and **Z** flags were defined as a test vector and the resulting recoded opcode was checked for correctness. Whether the correct source and destination registers have been extracted is also verified. Recall that the two branch control flags are **BRANCH**, which should be asserted if a branch is to occur because the opcode is a conditional branch and the corresponding flag causes the branch, and **NULL**, a flag which checks whether a branch offset is contained within the opcode for a short branch or whether it is contained in the next word in memory for a long branch. The proper setting of these two flags is also checked for by simulating several branch condition codes with several flag settings. The same set of test vectors was applied to the circuit implementation and the circuit edited until specifications were met.

4.3.2.2. Memory Facility Simulation

The memory facility is composed of the RIM and RIF units. The RAM model was generated by a program called CLASSIC. The model includes full timing verification, such as setup and hold timing as per the DPSRAM specs outlined previously. Test vectors for this model in the HOLIXZ program perform a write to two separate locations in the RAM and then read both locations back simultaneously to check that the data has been stored properly and is retrieved properly. The high output impedance provided by the memory block is also checked for. As the simulation model is computer generated, confidence in actual performance is quite high. In the real system, extensive test vectors should be generated to test for fabrication faults, such as two adjoining cells shorted together. There are standard test patterns, such as storing sequences of 1's alternated with 0's. The RIF interface unit was written using the FML2 asynchronous modeling language, whose functionality is verified by trying each function. The two possible inputs for address selection to Port A and Port B are each tried. Then the timing generation for a read and a write cycle for the RAM is verified. As described in the design section, the RIF was then manually synthesized into a circuit both from the RAM timing diagrams and from the behavioural model. A slight redesign was necessary at this stage due to behavioural model limitations. On the other hand, the actual circuitry had to break down an external clock into several states in order to properly generate these delays. Thus, the external clock (EXTCLK) had to be routed first to the RIF, then to the DVI where it is labelled INTCLK as previously in order to insert wait states, and then finally routed to the microcontrol unit.

4.3.2.3. Arithmetic Logic Unit Simulation

The ALU is composed of the ALUS and AIF blocks and was simulated both as a functional model written in the FML2 asynchronous model language, and then manually synthesized into a circuit, designated ALU. Both the ALUS and the ALU are tested with the same set of test vectors as they are functionally equivalent. The test vectors cycle the ALU through all of its functions, while testing all cases for the two flags for each function. For example, the SUB command subtracts two equal negative numbers to obtain zero with the zero flag set and the negative flag reset. A SUB command is then performed to obtain a negative number with the zero flag reset and the negative flag set. Then, a SUB command is performed to obtain a positive result with both flags reset. Finally, an alternating 1's and 0's pattern test is performed to catch any adjacent-bit errors. Similar test vectors are written

for the ADD, AND, OR, SWAP, and BTST commands. The AIF bus interface unit is also checked by cycling it through its functions. Thus, a check is performed to see that both the input multiplexer and the output hi-impedance bus interface work.

Once a functional model was obtained, it was translated into a circuit manually by using pre-defined blocks in the Standard Cell User-Designed Silicon SUDS library. That is to say, usage of standard library logic gates was made, rather than designing circuits from transistor level up. This has the advantage of saving much time and effort by using characterized and tested gates, but has the disadvantage of being limited by the extent of parts available and by their performance. This is quite similar to designing circuitry with discrete logic components. Cells available vary from simple, such as a 2-input NAND, to complex, such as a 16-bit multiplier. The ALU was designed using a hierarchical methodology, where the ALU's specifications and pinouts were known from the functional model, and the different functions were defined within the model. Thus, the ALU was subdivided into eight blocks. These are AND, OR, SWAP, ADDER, BTST, OUTPUT MUX, REGISTER A, and FLAG_LOGIC. These are straightforward and correspond easily to Figure 18 and Figure 19. On this diagram, most actual SUDS components are shown, such as MUX8, AND2, OR2, and MX2. A circuit schematic program called FUNDES was used, which links directly into the CMOS4 component library. First the high level eight unit block diagram was built, then each block was designed using components from the library. The ALU was then simulated until specifications were met by functional verification with the previously mentioned test vectors. These vectors had the buses changed into a series of bits because the actual chip tester hardware such as a GENRAD or SENTRY tester requires bit listings. The simulator thus used the CMOS4 component library for timing estimates of the circuit.

4.3.2.4. Fault Simulation of ALU

Fault simulation was also done on one of the blocks, namely the ALU. Fault simulation is a method to attempt to determine whether the test vectors will detect so-called faults or defects arising from fabrication defects such as pinholes in oxide and shorts between levels. Thus, gates can be modeled to be Stuck-at-1 (SA1) or Stuck-at-0 (SA0) in which the input or output of a gate is permanently connected to 1 or 0. The computer simulates this by generating a file of such faults. Simulation is then performed on a portion of possible SA1 or SA0 faults to determine whether the given set of test vectors will detect the faults. The more simulated faults, the higher the detection confidence level. The

percentage of faults detected by a test pattern is called its fault coverage. Although it is easy to simulate multiple simultaneous faults, it is difficult to determine which faults could be simultaneous. Thus, an assumption of single faults at a time is done. In general, a defective circuit may contain more than one fault or it may have other types of faults such as bridging faults. However, it is observed that simulation using single Stuck-At faults gives a fault coverage which is of the same order as in a real situation [30]. Several optimizations are commonly performed to reduce simulation costs during fault simulation. Fault collapsing reduces the amount of faults needing simulation by simplifying cases of faults. For example, a input SA0 to a NAND gate is the same as an output SA1 from this gate. Another example is that separate faults within a long chain of single fanin/single fanout gates cannot be distinguished. Another technique to lower simulation cost is the elimination of a faulty circuit from the simulation when that fault has been detected. This is referred to as "Fault Dropping".

Fault simulation was performed on the ALU block, and further test vectors were added to achieve high detection levels. As can be seen in appendix A (file ALUS FAULT), the given ALU test vectors "definitely detected" $97\% \pm 2\%$ of faults, "potentially detected" $3\% \pm 2\%$ of faults, had $0\% \pm 0\%$ "undetected" faults, and $0\% \pm 0\%$ of "dropped" faults, all this with a confidence level of 90%. Thus, design goals could set the minimum detected faults at 80% with a preferred goal of 100%. Unfortunately the higher the detection level, the more test vectors are needed and the more expensive test time is needed, driving chip cost up.

4.3.2.5. EPROM and Clock Simulation

For proper simulation of the system, the clock and external EPROM were needed. These were both written as FML2 asynchronous models and compiled using the FMLC compiler. The model for the EPROM contains the DTACK handshaking logic in addition to the actual EPROM contents. These are a series of opcodes representing a program to cycle the chip through various functions. Since the test program is fairly small, it was compiled by hand and modelled as a series of "case" statements switching on the EPROM address lines. The model has proper EPROM access delays but for simplicity does not check for proper setup and hold times, which are verified manually. The EPROM's test vectors verify the power-up program counter and stack pointer values stored in the EPROM, as well as the last EPROM memory location value. Finally, the EPROM high-impedance output and handshaking capabilities are checked versus specifications.

The clock model includes power-up reset signal generation logic. This block was verified simply by letting the model free-run and manually verifying the proper reset signal generation and constant clock signal output.

Since the EPROM and clock models were used to verify system functionality but are not part of the chip, no actual circuit was designed for them.

4.3.2.6. Bus Interface Controller

The bus interface controller (DVI) was written in the FML2 behavioural language as an asynchronous model. In its first implementation, the DVI was designed as a circuit, then written as an FML2 behavioural model for simulation. It included a special 8-bit EPROM interface with built-in programmable wait-state amount for access to this EPROM. This model was tested with a set of vectors as the other blocks, but it was later decided to implement the EPROM as a standard 16-bit eprom, unlike the microprocessor based system this chip is supposed to emulate. This is a deviation from the original design goal, but it was felt that designing a special interfacing for an EPROM is not realistic when wider EPROMs are now common. Also, it is felt that chip area would be better spent in incorporating other features such as a read buffer. As this new design is basically a modification and upgrade of the old DVI and since the circuit is fairly straightforward, the test vector generation stage was skipped and the DVI was debugged by observing its response during simulation of the entire chip. For actual implementation, however, this block should be tested separately from other units and treated as such with LSSD access to its boundaries.

4.3.2.7. MicroProgram Controller

Design of the microprogram controller MPC was done as an FML2 synchronous model. This is very similar to defining a state diagram with all signals defined and branching to the next state depending on several or no input variables. This design was done last, after all other units were implemented, allowing exact timing and signal definition specifications to be known as input to the design of this unit. Some minor upgrades were done on the system at this point to allow for proper actual microcode operation. The microcode was written and carefully analyzed before submitting it to the compiler and synthesizer, a very long and tedious process. Once synthesized by SYNFUL,

which takes several hours on the mainframe, an actual circuit is generated from the microcode by the mainframe, in this case as a Sea-of-Gates. At this point, the design was verified by connecting the MPC to the clock (CKC) and letting it free-run to observe whether the initialization control bits were being properly output. At this point, debugging of the system as a whole could begin.

4.3.2.8. System Simulation

The system is implemented as a macro file ALLM which joins together all of the pre-verified system components, including external components such as the clock and eeprom. Debugging and verification of the system was done manually by tracing signal generation as a combination of the microcode contained in the MPC and the program contained in the EPROM. As with all other components, many difficulties were encountered, most having to do with the simulator rather than with the design. Initially, units such as the bus interface controller DVI were simplified to their simplest state and other models modified to accommodate this. Then, as each unit started working, all the features were gradually brought back in order to have the full system simulation operating correctly. System simulation was considered complete when the chip did a proper power up sequence, including fetching the program counter (PC) and stack pointer (SP) from the EPROM and storing these in the internal DPSRAM. Then the program consisting of the series of opcodes forming a program shown in Table 18 had to execute perfectly. This completed chip functional verification. In an actual implementation, much more extensive module and system testing would have had to be done.

Step	Instruction	Expected Result
1	MOVE.W #AAAA, D3	D3 = \$AAAA, NF = 1, ZF = 0
2	MOVE.W #5555, D7	D7 = \$5555, NF = 0, ZF = 0
3	OR.W D7, D3	D7 = \$FFFF, NF = 1, ZF = 0
4	ADD.W D7, D3	D3 = \$AAA9, NF = 1, ZF = 0
5	BMI SKIP	Will Branch
6	ADD.W D7, D3	Skipped
SKIP: 7	CMP.W D7, D3	No Store, NF = 1, ZF = 0

Table 18, Program for System Simulation

4.3.3. Layout Area Estimates and Evaluation

Chip layout is mostly done using automatic layout software, such as BNR/CADENCE Place and Route or AUTOLAY2. The designer needs to specify the floor plan layout of the blocks and the synthesizer lays the chip out. It is important to keep functional blocks separate as they need to be debugged by isolating each block independently and using LSSD techniques. The layout software also generates a file for the digital simulator FUNSIM in order to simulate parasitic loading effects due to layout. Unlike chip design, which is usually done in functional block groups, it is often beneficial to do chip layout in a bit-slice organization. This reduces the need for wasteful signal buses by laying out the entire block for a single data bit. Then, this circuit is replicated for each data bit with control signals flowing across the bit slices in a perpendicular direction. Such a layout is optimal for a repetitive structure such as all the data bits within the ALU in this design.

Chip layout was not done except for a tentative layout of the ALU using AUTOLAY2. The layout should have been done on a Unix workstation with layout CAD software but these were not available to the author. Thus the mainframe had to be used and the package used to convert logic-block organization into bit-slice organization automatically, is only available on workstations, could not be used. As proper and full chip layout was not done, it was not possible to extract parasitic capacitances for re-simulation of the system as affected by the layout. The design was built with enough timing tolerances to ensure layout would not cause the design to fail.

Manual estimates of the area required for each block are shown in Table 19. The detailed calculations are shown in appendix B. The area estimate of 12 mm^2 is then doubled to 24 mm^2 to allow for routing and the addition of I/O pads, a chip with sides of less than 5 mm. By going to CMOS4S, a shrunken and fully compatible version of CMOS4, area could be reduced by over 35% to under 16 mm^2 with a chip side of under 4 mm. More recent technologies such as BatMOS could be used for even higher densities. Chip speed could also be accelerated. It is interesting to note the distribution of chip area for each function. About 70% of the area is used for the DPSRAM which is usable by programs and for registers; 20% is used for the microcontrol; and 10% for the ALU and other on chip functions. The number of transistors required for implementation of the designed chip, as shown in Table 20, are estimated to be 31,000 for the DPSRAM and 11,000 for the core, composed of control, ALU, and miscellaneous logic sections. The

core should thus be implemented as a "macro processor core" library cell, and the user should specify the desired amount of RAM for the specific application, with a minimum of 32 bytes required for register emulation. It may be beneficial to offer the user single port RAM as an option for memory expansion, as well as various smart co-processor units such as I/O drivers and communications ports. Although the chip size and proportions are good, two possible improvements could include investigating the impact on speed and area of using a PLA microcontrol rather than a sea-of-gates implementation in order to save area, and increasing functionality of the other modules at the cost of some area. Having such a large proportion of the area devoted to program usable memory is beneficial since it is directly usable by the user. On the other hand, it may be desirable to save more chip area to lower cost and this could easily be done by decreasing the amount of on-chip DPSRAM.

Block	Estimated Area mm ²	Estimated Area Portion of Total
AIF	0.056	0.5%
ALU	0.638	5.2%
BRA	0.148	1.2%
MPC	2.475	20.3%
RIF	0.063	0.5%
RAM (DPSRAM)	8.557	70.3%
DVI	0.229	1.9%
ENTIRE CHIP AREA	12 mm ²	100%
Technology	CMOS4	CMOS4S
Doubled for Routing and Pad I/O	24 mm ²	16 mm ²
Each Side of Chip	4.9 mm	4 mm

Table 19, Manual Estimate of Chip Area

Complexity	Number of Transistors
DPSRAM	31,000
Control, ALU, Logic	11,000
TOTAL	42,000

Table 20, Manual Estimate of Number of Transistors in Design

4.4. CONCLUSIONS ON THE DESIGN

The previous three chapters have dealt with various issues regarding the design of embedded instrumentation processors, providing the basis for the design undertaken in this thesis. First, a description of the application system referred to as the **BarTester** was provided and specifications for the design in this thesis listed. Design topics for embedded instrumentation processors such as RISC technology, bus interfaces, architectures, and pipelining were examined. Chip design methodologies were then discussed with examples shown. The issue of testability was overviewed, and finally, the necessity and methods of measuring value of a processor were examined. This foundation was then applied to design an embedded instrumentation processor core suitable for use as the processor in the **BarTester** system and compatible with the addition of on-chip peripheral digital units such as a camera interface and digital I/O.

The top-down and modular design of an embedded instrumentation processor meeting specifications was discussed. Performance attained has been shown to be of the same order required by the application. As specified in the introduction, the real-time sequencer was not designed, but the processor core's open architecture allows easy addition of such units. A small on-chip static RAM for data storage was incorporated as specified. An understanding of testability issues has been demonstrated by applying general concepts such as Fault Testing, test vector design, and a modular design. It was estimated the processor could be laid out onto a 24 mm^2 chip using CMOS4 technology. The area partitioning of 70% for the DPSRAM, of 20% for the microcontrol, and of 10% for the ALU and other on-chip functions was advantageous for such an embedded instrumentation processor. The size of 11,000 transistors for core functions and 31,000 transistors for memory functions suggests offering the core as a macro processor core cell, and letting the user specify the amount of DPSRAM to incorporate on-chip.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

5.1. THESIS SUMMARY AND CONCLUSIONS

Test instrumentation requires microprocessor based controllers as a bridge between analog electronics and large data collection and analysis systems. Other required functions may include control of mechanical components and processing of the acquired data. This thesis summarizes results of considerations for data and control path design in such an integrated embedded instrumentation processor. The application is a test system for semiconductor telecommunication lasers characterized in multi-laser bar form rather than in die form. Addressed processor core architecture design issues include RISC architectures, interrupt handling, embedded processor requirements, pipelining, parallelism, caches, and DSP features. Case studies on chip design methodologies have been presented. Testability issues such as scan methodologies and self test and diagnostic have been reviewed. Three methods for measurement of chip performance were presented. It was acknowledged that embedded processors are currently available on the market, however, this thesis focuses on a low-cost architecture designed to the gate level, and on requirements of specific features for embedded instrumentation control. A cost/feasibility analysis is needed to determine whether to implement the designed chip or whether a purchased processor with discrete logic would be more appropriate. An integrated processor core design for instrument control meeting all specifications has thus been developed. This design was simulated, the

layout area estimated, and the design evaluated, providing a basis for gate level synthesis of processor core design for instrumentation control.

It was shown that for integrated embedded instrumentation processors:

- RISC architectures are optimal for embedded instrument control applications, due to their low usage of chip real-estate, rapid response to interrupts, and the parallel execution of internal units.
- Chip area saved by using RISC may then be used to implement specific "smart" hardware units such as a coprocessor driven I/O, removing main processor real-time requirements. Strong interrupt processing, real-time I/O control, and communication capabilities are required.
- Small data and instruction caches are sufficient for embedded instrumentation software.
- DSP capability is desirable for activities such as data filtering, but not essential.
- Using a large DPSRAM to implement registers has a higher memory density than discrete register implementations. Chip area saved may then be used to implement numerous windowing DPSRAM registers for high-level language support with the effect of low interrupt response latency.
- Together, the "smart" modules and the large set of registers allow real-time response and reduce requirements for processor core performance.

While reviewing chip design and evaluation methodologies, the following aspects were highlighted:

- Extensive analysis of actual application software code and of hardware requirements leads to design optimization
- Benefits such as cost and time-to-market of implementing specific features on-chip rather than implementing them in software or on a separate chip must be considered.
- The design must be modular, testable, and preferably include Built In Self-Test features.
- Performance metrics of the chip may be quantitative, measuring parameters such as data path cycle time and memory access time or using benchmarking programs, or they may be qualitative such as plotting a Kiviat graph.

The top-down and modular design of an embedded instrumentation processor meeting all specifications was presented. Performance attained has been shown to be of the same order as the BarTester system processor for this application. As specified in the introduction, the real-time sequencer was not designed, but the processor core has an open architecture allowing future upgrades. Thus, the sequencer could easily be implemented as one of the processor's internal modules, a state machine interfacing directly to the processor core. A small on-chip DPSRAM for data storage has been provided which could be used as data or instruction cache, or for register windowing. This semi-RISC processor core decreases both test system cost and complexity by integrating as much of the digital circuitry as possible onto a single chip. Although an understanding of testability issues has been demonstrated and general concepts applied, such as Fault Testing, test vector design, and design modularity, incorporation of full testability circuitry in the chip design was not within the scope of the thesis. Layout and fabrication of the design as a silicon device were discussed but not implemented as these again are outside the scope of the thesis. It was estimated, though, that the processor, including the 192 bytes of on-chip DPSRAM, could easily be implemented, in CMOS4 technology, onto a chip 5 mm a side. It was argued that the area partitioning of 70% for the DPSRAM, of 20% for the microcontrol, and of 10% for the ALU and other on-chip functions was advantageous for such an embedded instrumentation processor. The size of 11,000 transistors for core functions and 31,000 transistors for memory functions suggests offering the core as a macro processor core cell, and letting the user specify the amount of DPSRAM to incorporate on-chip. Other functions such as communications or I/O ports could be drawn from a standard cell library, and custom units custom designed by the chip designer.

5.2. FUTURE WORK

Actual implementation of the designed processor with full testability built in would confirm design validity and performance. Topics which would further those introduced in this thesis abound: a few are described here. With the development of the proposed embedded instrumentation processor core, the importance of fine-grained parallelism became evident. This would require support structures in order to efficiently use such capabilities. For example, data and opcodes must be brought into the processor at appropriate rates, dispatched internally along shared buses, and results stored without slowing execution of further codes. To design these features while retaining the low-cost format would pose quite a challenge and would increase processor value. Examination of more application software would better highlight requirements for embedded

instrumentation processors, including whether the opcodes provided here are indeed optimal and whether the addition of other on-chip functionality such as a barrel-shifter would be beneficial. The availability of lower cost RAM should affect both system and processor design. Exploratory work examining the effects of such technology improvements on processor architecture could lead to new highly efficient architectures. The addition of analog and DSP capabilities on-chip would allow further integration of embedded systems for instrument control onto a single chip for lower system cost.

REFERENCES

- [1] J. McLeod, "Tough Choices Ahead (RISC vs. CISC: Where the Market's Heading)", *Electronics*, May 1989, pp.70-78.
- [2] C. Purkiser and J. Kardach, "The Intel 376 Family for Embedded Processor Applications", *IEEE Micro*, June 1988, pp.10-26.
- [3] R. S. Piepho and W. S. Wu, "A Comparison of RISC Architectures", *IEEE Micro*, August 1989, pp.51-62.
- [4] Sun Microsystems, "A RISC Tutorial", *Sun Microsystems*, Vol.24, No.4, May 1988.
- [5] T. Tokumaru et al, "Design of a 32-Bit Microprocessor, TX1", *IEEE Journal of Solid-State Circuits*, August 1989, pp.938-944.
- [6] F. Anceau, "The Architecture of MicroProcessors", Addison-Wesley Publishing Company, 1986.
- [7] R. C. Hansen, "New Optimizations for PA-RISC compilers", *Hewlett-Packard Journal*, June 199, pp.15-23.
- [8] L. Kohn and N. Margulis, "Introducing the Intel i860 64-Bit Microprocessor", *IEEE Micro*, August 1989, pp.15-30.
- [9] R. R. Oehler and R. D. Groves, "IBM RISC System/6000 Processor Architecture", *IBM Journal of Research and Development*, January 1990, pp.23-36.
- [10] R. W. Edenfield et al, "The 68040 Processor: Part 1, Design and Implementation", *IEEE Micro*, February 1990, pp.66-78.
- [11] J. Child and D. Wilson, "RISC Champions Challenge Moto in Embedded Control", *Computer Design*, October 1991, pp.98-116.
- [12] T. Wynia, "RISC and CISC Processors Target Embedded Systems", *Electronic Design*, June 1991, pp.55-70.
- [13] M. L. Fucio and B. Ng, "Hardware Architecture Considerations in the WE32100 Chip Set", *IEEE Micro*, April 1986, pp.29-46.
- [14] J. L. Hennessy and N. P. Jouppi, "Computer Technology and Architecture: An Evolving Interaction", *Computer*, September 1991, pp.18-29.
- [15] T. L. Johnson, "The RISC/CISC Melting Pot", *Byte*, April 1987, pp.153-160.
- [16] J. Jelezensky et al, "The MC68332 Microcontroller", *IEEE Micro*, August 1989, pp.31-50.
- [17] H. S. Stone, "Introduction to Computer Architecture", Science Research Associates, 1975.

- [18] D. Wilson, "Second-generation RISC chips: Intel Intros Family of i860s While Moto Goes Superscalar", *Computer Design*, July 1991, pp.42-47.
- [19] H. S. Stone and J. Cocke, "Computer Architectures in the 1990s", *Computer*, September 1991, pp.30-38.
- [20] S. McGeady, "Embedding Superscalar RISC", *Electronic Engineering Times*, April 1991, p.42.
- [21] J. L. Hennessy and D. A. Patterson, "Computer Architecture A Quantitative Approach", Morgan-Kaufmann Publishers Inc., 1990.
- [22] D. Bursky, "Evolving DSP Chips Will do More", *Electronic Design*, December 1990, pp.51-63.
- [23] J. Child, "16-bit MCUs Advance Along Many Paths", *Computer Design*, September 1992, pp.127-132.
- [24] R. Weiss and J. A. Schofield, "EDN's 19th Annual μ P/ μ C Chip Directory", *EDN*, November 1992, pp.75-161.
- [25] R. W. Edenfield et al, "The 68040 Processor: Part 2, Memory Design and Chip Verification", *IEEE Micro*, June 1990, pp.22-35.
- [26] A. Khan, "MIPS RISC: an Overview of the Architecture", *Computer Design*, Vol.28, No.22, November 1989, pp.12-14.
- [27] C. A. Gleason et al, "VLSI Circuits for Low-end and Midrange PA-RISC Computers", *Hewlett-Packard Journal*, August 1992, pp.12-22.
- [28] R. J. Feugate, "Introduction to VLSI Testing", Prentice-Hall, 1988.
- [29] D. P. Siewiorek, C. G. Bell, and A. Newell, "Computer Structures Principles and Examples", McGraw-Hill Book Company, 1982.
- [30] Northern Telecom, Proprietary Northern Telecom CAD Manuals Release 4. 04. 00 - Release 5. 02. 00, Northern Telecom Ltd., 1989-1991.

BIBLIOGRAPHY

- [1] C. Melear, "The Design of the 88000 RISC Family", *IEEE Micro*, April 1989, pp.26-38.
- [2] G. R. Lang et al, "An Optimum Parallel Architecture for High-Speed Real-Time Digital Signal Processing", *Computer*, February 1988, pp.47-56.
- [3] N. Margulis, "The Intel 80860", *Byte*, December 1989, pp.333-340.
- [4] T. Baji et al, "A 20-ns CMOS Micro-DSP Core for Video-Signal Processing", *IEEE Journal of Solid-State Circuits*, Vol.23, No.5, October 1988, pp.1203-1211.
- [5] I. Dancea, "Dynamically Changing the Logical Behaviour of a Microcomputer Interface", *IEEE Micro*, April 1989, pp.39-51.
- [6] V. K. L. Huang et al, "The AT&T WE32200 Design Challenge", *IEEE Micro*, April 1989, pp.14-25.
- [7] K. Yamamoto et al, "A Front-End Processor for Modems", *IEEE Journal of Solid-State Circuits*, Vol.24, No.6, December 1989, pp.1634-1638.
- [8] Texas Instruments Inc., "Intelligent Timer, Data EEPROM Featured On-board Newest TMS370 Microcontroller", *FYI*, March 1990.
- [9] D. Perlmutter et al, "The 80387 and its Applications", *IEEE Micro*, August 1987, pp.42-57.
- [10] M. Johnson, "System Considerations in the Design of the AM29000", *IEEE Micro*, August 1987, pp.28-41.
- [11] C. B. Hunter, "Introduction to the Clipper Architecture", *IEEE Micro*, August 1987, pp.6-26.
- [12] B. Eichen, "NEC's μ PD77230 Digital Signal Processor", *IEEE Micro*, December 1986, pp.60-69.
- [13] J. P. Roesgen, "The ADSP-2100 DSP Microprocessor", *IEEE Micro*, December 1986, pp.49-59.
- [14] K. L. Kloker, "The Motorola DSP56000 Digital Signal Processor", *IEEE Micro*, December 1986, pp.29-48.
- [15] G. A. Frantz et al, "The Texas Instruments TMS320C25 Digital Signal Microcomputer", *IEEE Micro*, December 1986, pp.10-28.
- [16] R. Cates, "Processor Architecture Considerations for Embedded Controller Applications", *IEEE Micro*, June 1988, pp.28-38.
- [17] M. Homewood et al, "The IMS T800 Transputer", *IEEE Micro*, October 1987, pp.10-26.

- [18] K. Sakamura, "Architecture of the TRON VLSI CPU", IEEE Micro, April 1987, pp.17-31.
- [19] N. P. Jouppi and J. Y. F. Tang, "A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance", IEEE Journal of Solid-State Circuits, October 1989, pp.1348-1359.
- [20] H. Inayoshi et al, "Realization of Gmicro/200", IEEE Micro, April 1988, pp.12-21.
- [21] D. P. Ryan, "Intel's 80960: An Architecture Optimized for Embedded Control", IEEE Micro, June 1988, pp.63-76.
- [22] K. M. Gutttag et al, "The TMS34010: An Embedded Microprocessor", IEEE Micro, June 1988, pp.39-52.
- [23] D. Atallah, "A Highly Integrated Solution to Fast I/O Processing", High Performance Systems, September 1989, pp.84-89.
- [24] N. Margulis, "Single-Chip RISC CPU Eases System Design", High Performance Systems, September 1989, pp.34-44.
- [25] A. Khan, "What's the Best Way to Minimize Memory Traffic ?", High Performance Systems, September 1989, pp.59-67.
- [26] S. Chan and R. Horst, "Building Parallelism into the Instruction Pipeline", High Performance Systems, December 1989, pp.53-60.
- [27] B. Fuhrt, "A RISC Architecture with 2-size, Overlapping Register Windows", IEEE Micro, April 1988, pp.67-80.
- [28] M. Miyata et al, "The TX1 32 Bit Microprocessor: Performance Analysis and Debugging Support", IEEE Micro, April 1988, pp.37-46.
- [29] S. Kimura et al, "Implementation of the V60/V70 and Its FRM function", IEEE Micro, April 1988, pp.22-36.
- [30] M. Horowitz et al, "MIPS-X: a 20-MIPS Peak, 32-Bit Microprocessor with On-Chip Cache", IEEE Journal of Solid-State Circuits, Vol.SC-22, No.5, October 1987, pp.790-799.
- [31] A. D. Berenbaum et al, "CRISP: A Pipelined 32-bit Microprocessor with 13-kbit of Cache Memory", IEEE Journal of Solid-State Circuits, Vol.SC-22, No.5, October 1987, pp.776-782.
- [32] M. Forsyth, "A 32-bit VLSI CPU with 15-MIPS Peak Performance", IEEE Journal of Solid-State Circuits, Vol.SC-22, No.5, October 1987, pp.768-775.
- [33] H. H. Chao et al, "Micro/370: A 32-bit Single-Chip Microprocessor", IEEE Journal of Solid-State Circuits, Vol.SC-21, No.5, October 1986, pp.733-740.

- [34] F. J. V. Wijk et al, "A 2- μ m CMOS 8-MIPS Digital Signal Processor with Parallel Processing Capability", IEEE Journal of Solid-State Circuits, Vol.SC-21, No.5, October 1986, pp.750-765.
- [35] H. Kadota et al, "A 32-bit CMOS Microprocessor with On-Chip cache and TLB", IEEE Journal of Solid-State Circuits, Vol.SC-22, No.5, October 1987, pp.800-807.
- [36] W. Stalling, "Reduced Instruction Set Computer Architecture", Proceedings of the IEEE, Vol.76, No.1, January 1988, pp.38-55.
- [37] D. J. Lilja, "Reducing the Branch Penalty in Pipelined Processors", Computer, July 1988, pp.47-55.
- [38] DeRosa and Levy, "An Evaluation of Branch Architectures", Proceedings of the 14th Annual Symposium Computer Architecture, June 1987.
- [39] D. R. Ditzel and H. R. McLennan, "Branch Folding in the CRISP Microprocessor: Reducing Branch Delay to Zero", Proceedings of the 14th Annual Symposium Computer Architecture, June 1987, pp.2-9.
- [40] K. F. Lee and H. Smith, "Branch Prediction Strategies and Branch Target Buffer Design", Computer, January 1984, pp.6-22.
- [41] J. Henessey et al, "Hardware/Software Tradeoffs for Increased Performance", ACM, March 1982, pp.2-11.
- [42] B. L. Peuto and L. J. Shustek, "An Instruction Timing Model of CPU Performance", Proceedings of the 4th Annual Symposium Computer Architecture, March 1977, pp.165-178.
- [43] D. W. Wall, "Register Windows Versus Register Allocation", Proceedings of the ACM SIGPlan 1988 on Programming Languages, June 1988, pp.67-78.
- [44] D. A. Patterson and C. H. Sequin, "A VLSI RISC", Computer, September 1982, pp.8-20.
- [45] R. P. Colwell et al, "Computers, Complexity, Controversy", Computer, September 1985, pp.8-19.
- [46] W. Hollingsworth et al, "The Clipper Architecture: Instruction Set Architecture and Implementation", Communications of the ACM, February 1989, pp.200-219.
- [47] T. C. Cooper et al, "A Benchmark Comparison of 32-bit Microprocessors", IEEE Micro, August 1986, pp.53-58.
- [48] A. Lunde, "Empirical Evaluation of Some Features of Instruction Set Processor Architectures", Communications of the ACM, Vol.20, No.3, March 1977, pp.143-153.
- [49] N. S. Woo and H. Shin, "A Technology-adaptive Allocation of Functional Units and Connections", ACM/IEEE Design Automation Conference, June 1989, pp.602-605.

- [50] D. Grover et al, "Data Path Optimization in Modern Computers", International Journal of Electronics, Vol.53, No.1, July 1982, pp.17-24.
- [51] R. Wilson, "MIPS Rethinks RISC with Superpipelining", Computer Design, February 1991, pp.28-32.
- [52] Z. Blazek, "Preliminary Design of a Parallel SIC Architecture", Microprocessor and Microprogramming, March 1990, pp.103-108.
- [53] A. G. Carr, "Real-Time Digital Signal Processing: a Review of Applications and Technology", Computing & Control Engineering Journal, March 1990, pp.77-81.
- [54] N. Margulis, "i860 Microprocessor Internal Architecture", Microprocessors and Microsystems, Vol.14, No.2, March 1990, pp.89-96.
- [55] J. H. Crawford, "The i486 CPU: Executing Instructions in One Clock Cycle", IEEE Micro, February 1990, pp.27-36.
- [56] A. Willenz and P. Bourekas, "Choosing the Right RISC Architecture", High Performance Systems, March 1990, pp.42-48.
- [57] R. Wilson, "Motorola 68040 Challenges RISC Microprocessors Head-on", Computer Design, February 1990, pp.14-21.
- [58] A. Picco et al, "The ST18940/41: an Advanced Single-chip Digital Signal Processor", 1989 IEEE International Symposium on Circuits and Systems, May 1989, pp.1559-1562.
- [59] H. A. Bustamante et al, "Stanford Telecom VLSI Design of a Convolutional Decoder", 1989 IEEE Military Communications Conference, October 1989, pp.171-178.
- [60] K. Keutzer, "Three Competing Design Methodologies for ASICs: Architectural Synthesis, Logic Synthesis and Module Generation", 26th ACM/IEEE Design Automation Conference, June 1989, pp.308-313.
- [61] J. C. Lo et al, "Concurrent Error Detection in Arithmetic and Logical Operations Using Berger Codes", Proceedings of the 9th Symposium on Computer Architectures, September 1989, pp.233-240.
- [62] D. K. Hwang et al, "Evaluation of a Reconfigurable Architecture for Digital Beamforming Using the OODRA Workbench", 26th ACM/IEEE Design Automation Conference, June 1989, pp.614-617.
- [63] V. K. Jain et al, "Wafer Scale Architecture for an FFT Processor", 1989 IEEE International Symposium on Circuits and Systems, May 1989, pp.453-456.
- [64] M. A. d'Abreu and F. F. Yassa, "An Automated Approach to the Design of DSP Systems", 1989 IEEE International Symposium on Circuits and Systems, May 1989, pp.449-452.

- [65] W. P. Burleson and L. L. Scharf, "VLSI Design of Inner-product Computers Using Distributed Arithmetic", 1989 IEEE International Symposium on Circuits and Systems, May 1989, pp.158-161.
- [66] P. M. Lu et al, "Architecture of a VLSI MAP for Belmac-32 Microprocessor", Digest of Papers-Compcon Spring 83, February 1983, pp.213-217.
- [67] S. T. Flannagan, "Synchronization Reliability in CMOS Circuits", IEEE Journal of Solid-State Circuits, Vol.SC-20, No.4, August 1985, pp.880-882.
- [68] P. Koopman, "The WISC Concept", BYTE, April 1987, pp.187-194.
- [69] P. Robinson, "How much of a RISC?", BYTE, April 1987, pp.143-150.
- [70] M. Ackerman and G. Baum, "The Fairchild Clipper", BYTE, April 1987, pp.161-174.
- [71] R. Mateosian, "1984, The Year of the 32-bit Microprocessor", BYTE, January 1984, pp.134-150.
- [72] P. Wells, "The 80286 Microprocessor", BYTE, November 1984, p.231-241.
- [73] F. Groepler and J. Kennedy, "The MC68020 32-Bit Microprocessor", BYTE, November 1984, pp.159-174.
- [74] T. L. Johnson, "A Comparison of MC68000 Family Processors", BYTE, September 1986, pp.205-218.
- [75] J. Markoff, "RISC Chips", BYTE, November 1984, pp.191-206.
- [76] R. Weiss, "MIPS to Unwrap its R4000 RISC", Electronic Engineering Times, January 1991.
- [77] R. Wilson and S. Bakeer, "MIPS Makes a 64-bit March on the Market", Electronic Design, February 1991.
- [78] J. J. Farrel, "Architecture of the VL86C020 RISC", Proceedings On VLSI and Computer Peripherals, June 1989, pp.5.140-5.141.
- [79] R. W. Hartenstein et al, "MOM: Map Oriented Machine", Proceedings On VLSI and Computer Peripherals, June 1989, pp.5.7-5.9.
- [80] T. Nogi, "ADENA Computer III", Memorial Faculty of Engineering of Kyoto University, February 1989, pp.135-152.
- [81] L. Kohn, "Intel i860: Single Chip Micro with Supercomputer Architecture", Australian Electronics Engineering, August 1989, pp.67-69.
- [82] G. Radin, "The 801 Minicomputer", ACM, March 1982, pp.39-47.
- [83] H. Nakagawa et al, "A Josephson 4-bit Processor for a Prototype Computer", Ext. Abstracts of 1989 International Superconductor Elements Conference, June 1989, pp.387-390.

- [84] S. Steinkerchner et al, "A 75000 Gate Graphics Chip Designed in 10 weeks", *Electronic Product Design*, October 1989, pp.37-43.
- [85] B. Case, "Pipelined Processor Pushes Performance", *International Symposium on Computer Architecture and Digital Signal Processing*, October 1989, pp.62-72.
- [86] S. Khan, "Jazing up the AT", *Computer Systems Europe*, December 1989, pp.13-16.
- [87] Motorola Inc., "MC88100: 32-bit Third-Generation RISC Microprocessor", Motorola, June 1988.
- [88] J. Thompson, "Performance Enters Embedded MIPS Fray", *Electronic Engineering Times*, August 1991, p.16.
- [89] T. Williams, "Performance Pushes RISC Chips Into Real-time Roles", *Computer Design*, September 1991, pp.79-86.
- [90] W. P. Birmingham et al, "The Micon System for Computer Design", *IEEE Micro*, October 1989, pp.61-67.
- [91] D. Wilson, "New 29000s Promise Faster Laser Printer Design", *Computer Design*, July 1991, pp.48-50.
- [92] R. Wilson, "Sun Unveils Viking Superscalar Design", *Electronic Engineering Times*, September 1991, p.1, p.53, p.56.
- [93] C. P. Ravikumar et al, "Parallel Circuit Partitioning on a Reduced Array Architecture", *Computer Aided Design*, September 1989, pp.447-455.
- [94] D. Bursky, "Multimillion-Transistor Budget Delivers Supercomputer-like Chip", *Electronic Design*, June 1991, p.31-34.
- [95] L. Ya-min et al, "A Hardware Implementation of SPARC-compatible RISC on Bit-slice Chips", *1989 International Symposium of Computer Architectures and Digital Signal Processing*, October 1989, pp.177-179.
- [96] T. Williams, "RISC Instruction Benchmarks Spark Performance Debate", *Computer Design*, May 1991, pp. 69-72.
- [97] H. Tenhuren et al, "VLSI Design Methodologies for Digital Signal Processing", *17th Yugoslav Conference on Microelectronic Processes*, June 1989, pp.246-251.
- [98] T. Kiuchi et al, "Digital signal processor for Audio Application", *NEC Research and Development*, July 1989, pp.37-42.
- [99] T. Hayashi et al, "A 5.6MIPS Call-handling Processor for Switching Systems", *IEEE Journal of Solid-State Circuits*, August 1989, pp.945-950.
- [100] K. Sawada et al, "A 32-kbyte Integrated Cache Memory", *IEEE Journal of Solid-State Circuits*, August 1989, pp.881-888.

- [101] D. Wilson, "Are Secondary Caches for the 486 Worth the Money?", *Computer Design*, May 1991, pp.28-32.
- [102] J. Gessner et al, "Synthesis of Control Units in a Design Environment for Chip Architectures", *Microprocessors and Microprogramming*, June 1989, pp.465-471.
- [103] R. M. King et al, "A Special Purpose Coprocessor Supporting Cell Placement and Floorplanning", *Proceedings of the IEEE 1989 Custom IC Conference*, June 1989, pp.3.1.1-3.1.4.
- [104] R. Hum, "A Major Innovation from BNR - Chips With Self-testing Circuitry", *Northern Telecom*, Vol.2, Is.10, March 1991, pp.1-2.
- [105] S. M. Mahmud, "Communication Performance in Hierarchical-bus Systems", *ISCAS 1989*, June 1989, pp.122-125.
- [106] J. C. Michalina, "Application Specific DSP at SGS Thomson", *ISCAS 1989*, June 1989, pp.627-630.
- [107] K. K. Parhi, "Nibble-Serial Arithmetic Processor Designs via Unfolding", *ISCAS 1989*, June 1989, pp.635-640.
- [108] W. Bao and M. A. Bayoumi, "Systolic Architectures for High Order Correlation Neural Nets", *ISCAS 1989*, June 1989, pp.1199-1202.
- [109] W. Li, I. L. Sidharta, and H. S. Fetterman, "A Computation Engine for the FFT/FCT/FHT Using AT&T DSP32", *ISCAS 1989*, June 1989, pp.1272-1275.
- [110] G. A. Jullien and W. C. Miller, "Homogeneous VLSI for 2-D Digital Signal Processing", *ISCAS 1989*, June 1989, pp.1388-1391.
- [111] A. R. Rostampour and D. Soltis, "The Design of Symbiotic Scalar-Systolic Array Processor SSSAP", *ISCAS 1989*, June 1989, pp.1392-1395.
- [112] C. C. Chakrabarti and J. JaJa, "Optimal Architectures for Two Dimensional Transforms", *ISCAS 1989*, June 1989, pp.1495-1498.
- [113] T. Stouraitis, "An Efficient VLSI Implementation of Logarithmic Signal Processors", *ISCAS 1989*, June 1989, pp.1540-1551.
- [114] Y. C. Lim and B. Liu, "Pipelined Implementation of Recursive Filter", *ISCAS 1989*, June 1989.
- [115] R. Wilson, "CPUs Keep Clashing: CISC Keeps Coming", *Electronic Engineering Times*, Is.715, October 1992, p.1, p.82.
- [116] W. Handler, "The Impact of Classification Schemes on Computer Architecture", *Proceedings of the 1977 International Conference on Parallel Processing*, June 1986, pp.3-11.
- [117] J. B. Dennis, "The Varieties of Data Flow Computers", *Proceedings of the 1st International Conference on Distributed Computer Systems*, October 1979, pp.51-60.

- [118] D. P. Agarwal, "High-speed Arithmetic Arrays", IEEE Transactions on Computers, March 1979, pp.83-92.
- [119] H. J. Sips, "Bit-sequential Arithmetic for Parallel Processors", IEEE Transactions on Computers, January 1984, pp.93-106.
- [120] T. N. Mudge et al, "Analysis of Multiple Bus Interconnection Networks", Proceedings of the 1984 International Conference on Parallel Processing, June 1984, pp.155-159.
- [121] F. A. Briggs and M. Dubois, "Effectiveness of Private Caches in Multiprocessor Systems with Parallel-Pipelined Memories", IEEE Transactions on Computers, January 1983, pp.165-176.
- [122] R. B. Lee and A. L. Wiemann, "New Design Methodologies and Circuits Needed for Parallel VLSI Supercomputers", International Conference on Circuits and Computers, June 1982, pp.285-292.
- [123] H. T. Kung, "Why Systolic Architectures?", Computer, January 1982, pp.300-309.
- [124] H. D. Toong and A. Gupta, "An Architectural Comparison of Contemporary 16-bit Microprocessors", IEEE Micro, Vol.1, May 1981, pp.26-37.
- [125] P. Heller et al, "Memory Protection Moves onto 16-bit Microprocessor Chip", Electronics, Vol.55, February 1982, pp.133-137.
- [126] D. Tjoa and T. Rossi, "A Task Driven Peripheral Interface", Wescon Conference Records, June 1981, pp.54-59.
- [127] S. Bal et al, "The NS16000 Family - Advances in Architecture and Hardware", IEEE Computer, June 1982, pp.83-92.
- [128] R. V. Orlando and T. L. Anderson, "An Overview of the 9900 Microprocessor Family", IEEE Micro, Vol.1, August 1981, pp.38-44.
- [129] R. Ochester, "Low-cost 16-bit Microprocessor has Performance of Midrange Computer", Electronics, Vol.54, November 1981, pp.129-133.
- [130] A. Gupta and D. Toong, "An Architectural Comparison of 32-bit Microprocessors", IEEE Micro, Vol.3, February 1983, pp.9-22.
- [131] H. Jacob, "An Architecture for the 80's - the Intel iAPX 432", Midcon Conference Records, June 1981, pp.169-174.
- [132] D. L. Budde et al, "The Execution Unit for the VLSI432 General Data Processor", IEEE Journal of Solid-State Circuits, Vol.SC-16, October 1981, pp.514-521.
- [133] A. Kaminker et al, "A 32-bit Microprocessor with Virtual Memory Support", IEEE Journal of Solid-State Circuits, Vol.SC-16, October 1981, pp.548-557.
- [134] C. Davis et al, "Gate Array Embodies System/370 Processor", Electronics, Vol.53, October 1980, pp.140-143.

- [135] A. Gupta and H. D. Toong, "Performance Evaluation", *Advanced Microprocessors*, June 1983, pp.219-220.
- [136] A. Gupta and H. D. Gupta, "Evaluation Kernals for Microprocessor Performance Evaluation Analyses", *Performance Evaluation*, Vol.2, May 1982, pp.1-8.
- [137] R. K. Bell et al, "The Big Three - Today's 16-bit Microprocessor", *Proceeding of the 13th Annual Microprogramming Workshop*, June 1980, pp.126-138.
- [138] H. G. Cragon, "The Elements of Single-chip Microcomputer Architecture", *IEEE Computer*, Vol.13, October 1980, pp.27-30, pp.32-41.
- [139] D. A. Patterson and C. H. Sequin, "Design Considerations for Single-chip Computer of the Future", *IEEE Transactions on Computers*, Vol.C-29, February 1980, pp.269-276.
- [140] J. P. Hayes, "A Survey of Bit-sliced Computer Design", *Journal of Digital Systems*, September 1981, pp.277-300.
- [141] H. Watanabe, "A VLSI Fuzzy Logic Controller with Reconfigurable, Cascadable Architecture", *IEEE Journal of Solid-State Circuits*, Vol.25, No.2, April 1990, pp.376-382.
- [142] R. Goering, "Is RISC Ready for Real-time Applications?", *High Performance Systems*, March 1990, pp.19-29.
- [143] R. Goering, "RISC Processors", *High Performance Systems*, March 1990, pp.30-41.
- [144] A. Willenz and P. Bourekas, "Choosing the Right RISC Architecture", *High Performance Systems*, March 1990, pp.42-48.
- [145] W. D. Gardner, "Neural Nets Get Practical", *High Performance Systems*, March 1990, pp.68-72.
- [146] J. P. Murray, "The Trade-offs in Neural-net Implementations", *High Performance Systems*, March 1990, pp.74-78.
- [147] R. D. Reutberg et al, "The Monarch Parallel Processor Hardware Design", *Computer*, April 1990, pp.18-30.
- [148] J. H. Moreno and T. Lang, "Matrix Computations on Systolic-type Meshes: An Introduction to the Multimesh Graph Method", *Computer*, April 1990, pp.32-51.
- [149] P. Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors", *Computer*, June 1990, pp.12-24.
- [150] P. J. Teller, "Translation-lookaside Buffer Consistency", *Computer*, June 1990, pp.26-36.
- [151] V. P. Nelson, "Fault-Tolerant Computing: Fundamental Concepts", *Computer*, July 1990, pp.19-25.

- [152] J. A. Mills, "A Pragmatic View of the System Architect", *Communications of the ACM*, Vol.28, No.7, July 1985, pp.708-717.
- [153] R. W. Priester et al, "Signal Processing with Systolic Arrays", *International Conference Parallel Processing*, June 1981, pp.207-215.
- [154] H. Li, "A VLSI Modular Architecture Methodology for Real-time Signal Processing Applications", *Proceeding of the 1983 International Conference Parallel Processing*, June 1983, pp.319-324.
- [155] E. H. Wold and A. M. Despain, "Pipeline and Parallel-pipeline FFT Processors for VLSI Implementation", *IEEE Transactions on Computers*, Vol.C-33, No.5, May 1984, pp.414-426.
- [156] S. L. Tanimoto, "A Pyramidal Approach to Parallel Processing", *Proceedings of the 10th Annual Symposium Computer Architectures*, June 1983, pp.372-378.
- [157] D. A. Moon, "Architecture of the Sybolics 3600", *Proceedings of the 12th Annual Symposium Computer Architectures*, June 1985, pp.76-83.
- [158] R. Bisiani et al, "Task-oriented Architectures", *Proceedings of the IEEE*, Vol.71, No.7, July 1983, pp.885-898.
- [159] S. Uchida, "Inference Machine: From Sequential to Parallel", *IEEE Computer Society*, June 1983, pp.410-416.
- [160] Y. Ishikawa et al, "The Design of an Object Oriented Architecture", *Proceedings of the 11th Annual Symposium Computer Architectures*, June 1984, pp.178-187.
- [161] C. Ebeling and A. Palay, "The Design and Implementation of a VLSI Chess Move Generator", *Proceedings of the 11th Annual Symposium Computer Architectures*, June 1984, pp.74-80.
- [162] H. Kameko et al, "Realizing the V80 and its System Support Functions", *IEEE Micro*, April 1990, pp.56-69.
- [163] H. Hidaka et al, "The Cache DRAM Architecture: a DRAM with On-chip Cache Memory", *IEEE Micro*, April 1990, pp.14-25.
- [164] T. Kitahara et al, "The Gmicro/300 32 Bit Microprocessor", *IEEE Micro*, June 1990, pp.68-75.
- [165] E. W. Brown et al, "Implementing Sparc in ECL", *IEEE Micro*, February 1990, pp.10-22.
- [166] M. Birman et al, "Developing the WTL3170/3171 Sparc Floating Point Coprocessors", *IEEE Micro*, February 1990, pp.55-64.
- [167] D. Bursky, "Pipelined CISC Processor Exceeds RISC Throughput", *Electronic Design*, January 1990, pp.91-93.

- [168] D. Bursky, "Floating Point Math Chip Delivers 200MFlops Peak", *Electronic Design*, February 1990, pp.51-55.
- [169] J. Gosh, "Risky Business: Where CISC Meets RISC", *Electronics*, February 1990, pp.74-76.
- [170] D. Bursky, "ISSCC Digital Technology", *Electronic Design*, February 1990, pp.63-82.
- [171] G. Bunza et al, "Design Automation in the 1990's: Driven to Abstraction", *Electronic Design*, January 1990, pp.91-97.
- [172] D. Bursky, "Digital IC's in the 1990's: Nearly Unlimited On-chip Resources", *Electronic Design*, January 1990, pp.79-89.
- [173] G. F. Grohoski, "Machine Organization of the IBM RISC System/6000 Processor", *IBM Journal of Research and Development*, Vol.34, No.1, January 1990, pp.37-58.
- [174] R. K. Montoye et al, "Design of the IBM RISC System/6000 Processor Floating-point Execution Unit", *IBM Journal of Research and Development*, Vol.34, No.1, January 1990, pp.59-70.
- [175] E. Hokenek and R. K. Montoye, "Leading Zero Anticipator (LZA) in the IBM RISC System/6000 FP Execution Unit", *IBM Journal of Research and Development*, Vol.34, No.1, January 1990, pp.71-77.
- [176] I. M. Ratiu and H. B. Bakoglu, "Pseudorandom Built-in Self-test Methodology and Implementation for the IBM RISC System/6000 Processor", *IBM Journal of Research and Development*, Vol.24, No.1, January 1990, pp.78-84.
- [177] H. S. Warren Jr., "Instruction Scheduling for the IBM RISC System/6000 Processor", *IBM Journal of Research and Development*, Vol.34, No.1, January 1990, pp.85-92.
- [178] M. C. Gohumbic et al, "Instruction Scheduling Beyond Basic Blocks", *IBM Journal of Research and Development*, Vol.34, No.1, January 1990, pp.93-97.
- [179] M. Alsup, "Motorola's 88000 Family Architecture", *IEEE Micro*, June 1990, pp.48-66.
- [180] R. Wilson, "Controllers Ride 16-bit Wave", *Electronics Engineering Times*, March 1992, pp.53-54.
- [181] D. Lammers, "NEC Fields Low-power Embedded RISC", *Electronics Engineering Times*, July 1992.
- [182] R. Goering and S. Runyon, "Mentor Bundles Top-down Tools", *Electronics Engineering Times*, October 1992, p.10.
- [183] Texas Instruments Inc., "Fixed Point DSP Family Expands", *FYI*, Vol.7, Is.8, October 1990, pp.2-3.

- [184] Texas Instruments Inc., "First Parallel Processing DSP disclosed", FYI, Vol.7, Iss.8, October 1990, p.1.
- [185] Intel Inc., "A New Member of the i960 Processor Family", MicroComputer Solutions, November 1990, p.10.
- [186] Texas Instruments Inc., "TMS370 Configurable μ Cs Product Bulletin", TI Product Bulletin, May 1989.
- [187] Texas Instruments Inc., "Funny, It Doesn't Look Like a Microcontroller", Texas Instruments, October 1990.
- [188] J. Thompson, "The Embedded RISC War Zone", Electronic Engineering Times, November 1990, p.1, p.100.
- [189] Texas Instruments Inc., "Parallel-Processing Floating-Point DSP", TI Product Advertisement, August 1990.
- [190] Northern Telecom Inc., "Silicon Gate & ASIC Development Process", NTE Mainframe, December 1990.
- [191] R. Woolnough, "Philips Builds on MCU Work", Electronic Engineering Times, November 1990.
- [192] T. Costlow, "Atmel Plans MISC Chips", Electronics, December 1990, pp.60-61.
- [193] L. T. Wang and S. Mourad, "SST: Scan Self-test for Sequential Machines", IEE Proceedings Part E, Vol.136, No.6, November 1989, pp.560-574.
- [194] F. Hwang, "Built-in Testability Improves Auto IC's", Test & Measurement World, February 1990, pp.41-47.
- [195] S. F. Scheiber, "Reducing Simulation Efforts for Complex Devices", Test & Measurement World, January 1990, pp.69-72.
- [196] B. Gupta et al, "On Designing Robust Testable CMOS Combinational Circuits", IEE Proceedings Part E, Vol.136, No.4, July 1989, pp.329-338.
- [197] J. H. Aylor and B. W. Johnson, "Structured Design for Testability in Semicustom VLSI", IEEE Micro, February 1986, pp.51-58.
- [198] M. A. Bayoumi and C. H. Yang, "Reconfigurable Testable Bit-serial Multiplier for DSP Applications", IEE Proceedings Part E, Vol.136, No.6, November 1989, pp.517-523.
- [199] H. Fujiwara, "A Design of Programmable Logic Arrays with Random Pattern-testability", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.5-10.
- [200] M. Katoozi and M. Soma, "A Testable CMOS Synchronous Counter", IEEE Journal of Solid-State Circuits, Vol.23, No.5, October 1988, pp.1241-1248.

- [201] S. Sastry, "Detectability of CMOS Stuck-open Faults Using Random and Pseudorandom Test Sequences", IEEE Transactions on CAD, Vol.7, No.9, September 1988, pp.933-946.
- [202] D. L. Liu and E. J. McCluskey, "Design of Large Embedded CMOS PLA's for Built-in Self-test", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.50-59.
- [203] L. Carter et al, "TRIM: Testability Range by Ignoring the Memory", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.38-49.
- [204] S. W. Chan and C. L. Wey, "The Design of Concurrent Error Diagnosable Systolic Arrays for Band Matrix Multiplications", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.21-37.
- [205] P. Mazumder et al, "Methodology for Testing Embedded Content Addressable Memories", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.11-20.
- [206] L. T. Wang and E. J. McCluskey, "Hybrid Designs Generating Maximum-length Sequences", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.91-99.
- [207] K. Iwasaki, "Analysis and Proposal of Signature Circuits for LSI Testing", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.84-90.
- [208] T. W. Williams et al, "Bounds and Analysis of Aliasing Errors in Linear Feedback Shift Registers", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.75-83.
- [209] E. J. McCluskey et al, "Probability Models for Pseudorandom Test Sequences", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.68-74.
- [210] S. J. Upadhyaya and K. K. Saluja, "A New Approach to the Design of Built-in Self-test PLA's for High Fault Coverage", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.60-67.
- [211] M. Abadir et al, "Logic Design Verification via Test Generation", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.138-148.
- [212] M. H. Schulz et al, "SOCRATES: a Highly Efficient Automatic Test Pattern Generation System", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.126-137.
- [213] R. J. Cosentino, "Concurrent Error Correction in Systolic Arrays", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.117-125.
- [214] N. K. Jha, "Testing for Multiple Faults in Domino-CMOS Logic Circuits", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.109-116.
- [215] J. L. A. Hughes, "Multiple Fault Detection Using Single Fault Test Sets", IEEE Transactions on CAD, Vol.7, No.1, January 1988, pp.100-108.

- [216] D. L. Landis, "A Self-test System Architecture for Reconfigurable WSI", International Test Conference 1989, June 1989, pp.275-282.
- [217] J. Turino, "Interfacing to Boundary Scan Chips for System Level BIT", Autotestcon 1989 Conference Record, June 1989, pp.310-313.
- [218] R. Goering, "Synopsys Tools Pioneer Design Reuse", Electronics Engineering Times, September 1992, p.4, p.58.
- [219] N. Vasanthavada et al, "A New Built-in Test Scheme for DVCS Circuits", ISCAS 1989, June 1989, pp.375-378.
- [220] D. M. Wu and E. P. Hsieh, "CMOS Stuck-open Fault Modeling, Detection, Simulation", ISCAS 1989, June 1989, pp.379-383.
- [221] M. Katoozi and M. Soma, "Practical Built-in Test of CMOS State Machines With Realistic Faults", ISCAS 1989, June 1989, pp.384-387.
- [222] M. Tremblay and Y. Tamir, "Support for Fault-tolerance in VLSI Procesors", ISCAS 1989, June 1989, pp.388-393.
- [223] J. H. Kim and P. Rhee, "Reconfiguration Algorithm of Fault-tolerant Two-dimensional VLSI Array", ISCAS 1989, June 1989, pp.394-397.
- [224] C. Brown, "Supercomputing Comes to the Desktop", Computing 2000, June 1992, pp.C22-C25.
- [225] I. Koren and A. D. Singh, "Fault Tolerance in VLSI circuits", Computer, July 1990, pp.73-83.
- [226] T. Markas et al, "On Distributed Fault Simulation", Computer, January 1990, pp.40-52.
- [227] E. J. McCluskey, "Design Techniques for Testable Embedded Error Checkers", Computer, July 1990, pp.84-88.
- [228] M. Franklin et al, "A Built-in Self-Test Algorithm for Row/Column Pattern Sensitive Faults in RAMs", IEEE Journal of Solid-State Circuits, Vol.25, No.2, April 1990, pp.514-524.
- [229] M. Katoozi and M. Soma, "Built-in Test of CMOS State Machines with Realistic Faults: a System Perspective", IEEE Journal of Solid-State Circuits, Vol.25, No.2, April 1990, pp.482-489.
- [230] A. Miczo and C. Miller, "Test Generator Extracts Design Intent From Behavioral Models", High Performance Systems, May 1990, pp.54-66.
- [231] C. Ellingham, "The New Challenges of System Testing", Test, December 1989/January 1990, pp.8-11.

- [232] M. E. Levitt and J. A. Abraham, "Physical Design of Testable VLSI: Techniques and Experiments", IEEE Journal of Solid State Circuits, Vol.25, No.2, April 1990, pp.474-481.
- [233] S. Patil and P. Banerjee, "A Parallel Branch and Bound Algorithm for Test Generation", IEEE Transactions on CAD IC Systems, Vol.9, No.3, March 1990, pp.313-322.
- [234] K. Kuchcinski and Z. Peng, "Testability Analysis in a VLSI High-level Synthesis System", Microprocessors and Microprogramming, March 1990, pp.295-300.
- [235] S. Rangarajan et al, "Built-in Testing of Integrated Circuit Wafers", IEEE Transactions on Computers, Vol.39, No.2, February 1990, pp.195-205.
- [236] T. Williams, "Siemens Intros New Generations of 40-MHz 8- and 16-bit MCUs", Computer Design, July 1992, p.108.
- [237] S. H. Choi, "Testing of Data Paths in VLSI Arrays", IEE Proceedings Part E, Vol.137, No. 2, March 1990, pp.154-158.
- [238] G. Liu and Y. F. Tsao, "Computer-aided Post-silicon Verification Ventures Beyond Design, Test", Electronic design, January 1990, pp.159-166.
- [239] A. C. Erdal and P. A. Uszynski, "A Global Chip Test Implementation Including Built-in Self-test", Proceedings of the 1988 IEEE International Conference on Computer Design, October 1988, pp.446-449.
- [240] F. Siavoshi, "WTPGA: a Novel Weighted Test-pattern Generation Approach for VLSI Built-in Self Test", International Test Conference 1988 Proceedings, September 1988, pp.258-262.
- [241] D. L. Liu and E. J. McCluskey, "Design of Large Embedded CMOS PLAs for Built-in Self Test", Proceedings of the 1987 IEEE International Conference on Computer Design, October 1987, pp.678-681.
- [242] D. L. Landis, W. A. Check, and D. C. Muha, "Influence of Built-in Self-test on the Performance of Fault Tolerant VLSI Multiprocessors", Proceedings of the 1987 International Conference on Parallel Processors, August 1987, pp.114-116.
- [243] V. Blaschke et al, "Modular Built-in Testprocessor for VLSI-chips", IEEE Computer Society Press, May 1987, pp.480-481.
- [244] T. W. Williams, "Built-in-self-testing", Proceedings of the VLSI and Computer 1st International Conference, May 1987, pp.345-348.
- [245] R. Wilson, "Fujitsu Extends Sparclite", Electronic Engineering Times, August 1992.
- [246] C. W. Starke, "An Efficient Self-test Strategy for Testing VLSI Chips", Proceedings on VLSI and Computer Peripherals, May 1989, pp.5.116-5.119.

- [247] P. Camurati et al, "ESTA: an Expert System for DFT Rule Verification", IEEE Transactions on CAD IC Systems, Vol.7, No.11, November 1988, pp.1172-1180.
- [248] S. M. I. Adhem et al, "An M1-based Expert System LSSD Rules Checker for VLSI Circuits", IEEE MONTECH 1987 Conference :COMPINT, November 1987.
- [249] S. N. Bhatt et al, "Partitioning Circuits for Improved Testability", Advanced Research in VLSI 4th MIT Conference, April 1986, pp.91-106.
- [250] J. H. Aylor et al, "Structured Design for Testability in Semicustom VLSI", IEEE Micro, February 1986, pp.51-58.
- [251] A. Quirt, "Computer Architectures for Event Processing", Pic/BNR, January 1990.
- [252] E. K. Vida-Torku et al, "Test Generation for VLSI Chips With Embedded Memories", IBM Journal of Research and Development, Vol.24, No.2/3, March/May 1990, pp.276-287.
- [253] R. W. Bassett et al, "Boundary Scan Design Principles for Efficient LSSD ASIC Testing", IBM Journal of Research and Development, Vol.24, No.2/3, March/May 1990, pp.355-362.
- [254] C. W. Starke, "Design for Testability and Diagnosis in a VLSI CMOS System/370 Processor", IBM Journal of Research and Development, May 1990.
- [255] B. L. Keller and T. J. Snethen, "Built-in Self-test Support in IBM Engineering Design System", IBM Journal of Research and Development, Vol.24, No.2/3, March/May 1990, pp.406-415.
- [256] J. Novellino, "ITC 1992 Integrates Design and Test", Electronic Design, September 1992, pp.65-68.
- [257] L. Wirbel, "AMD29205 Gets 16-bit I/O", Electronic Engineering Times, September 1992.
- [258] S. Runyon, "Design for Testability Outshines ATE", Electronic Engineering Times, September 1992.
- [259] S. F. Scheiber, "Design-for-Testability Comes of Age", Test & Measurement World, November 1990, pp.85-95.
- [260] Northern Telecom Inc., "Test Vector Review", NTE Mainframe, December 1990.
- [261] J. G. McWhirter and J. V. McCanny, "VLSI Technology & Design", Academic Press, 1987.
- [262] IEEE Standards Board, "IEEE Standard Test Access Port and Boundary-scan Architecture", The Institute of Electrical and Electronics Engineers, 1990.
- [263] E. E. Swartzlander Jr., "VLSI Signal Processing Systems", Kluwer Academic Publishers, 1986.

- [264] D. Wilson, "RISC Processors Dressed Up for Embedded Applications", *Computer Design*, December 1991, pp.34-39.
- [265] R. Wilson, "How Chips Developed the Super386", *Electronic Engineering Times*, October 1991, pp.93-97.
- [266] A. Samad, "Making the Best of BIST", *Electronic Engineering Times*, October 1991, p.67-68.
- [267] R. Goering, "Test Synthesis Gets Passing Grades", *Electronic Engineering Times*, October 1991, pp.45-51.
- [268] D. Wilson, "Rethinking Signal Processing Philosophy", *Computer Design*, January 1992, pp.112-114.
- [269] E. J. McCluskey, "VLSI Design for Testability", *Center for Reliable Computing*, 1984.
- [270] D. L. Liu, "Testable Structures for CMOS VLSI Circuits", *Center for Reliable Computing*, 1987.
- [271] P. Bardell, W. H. AMcAnney, and J. Savir, "Built-in Test for VLSI: Pseudorandom Techniques", *John Wiley & Sons*, 1987.
- [272] F. F. Tsui, "LSI/VLSI Testability Design", *McGraw-Hill Book Company*, 1987.
- [273] A. S. Tanenbaum, "Structured Computer Architecture", *Prentice-Hall*, 1976.
- [274] Texas Instruments Inc., "TMS370CX32 8-bit MicroControllers", *Texas Instruments*, February 1990.
- [275] Motorola Inc., "MC68000 8-/16-/32-bit Microprocessors Programmer's Reference Manual", *Prentice-Hall*, 1986.
- [276] J. Child, "DSP Offers More Cash for the Cash", *Computer Design*, January 1992, p.111.
- [277] D. Johnson and S. Davidson, "Partial Scans Have Pitfalls", *Electronic Engineering Times*, September 1992, p.40.
- [278] G. vanWauwe and J. vanSas, "DFT has BIST in Full Scan", *Electronic Engineering Times*, September 1992, p.40.
- [279] M. Myers, "Boundary Scan Beefs up Testability", *Electronic Engineering Times*, September 1992, p.44.
- [280] D. Curtis, "Quiescent Current Assures Tested Chip", *Electronic Engineering Times*, September 1992, p.44.
- [281] S. Beal, "FPGAs Facilitate Test-vector Generation", *Electronic Engineering Times*, September 1992.
- [282] J. Moutafis, "Swatch IC Unit Builds Low-Power RISC Core", *Electronic Engineering Times*, November 1992, p.10.

- [283] D. Wilson, "DSP Functions Keep Moving Onto Microcontrollers", *Computer Design*, October 1991, pp.40-41.
- [284] D. J. Wharton, "Efficient Test Methods: the Choice is Yours", *Electronic Engineering Times*, January 1992, p.42.
- [285] R. Tullos and C. Yau, "System DFT Calls for Linking New with Old", *Electronic Engineering Times*, January 1992, p.40.
- [286] J. P. Braun and M. Sherred, "Chips' Super386: DFT Gets Put to the Test", *Electronic Engineering Times*, January 1992, pp.44-45.
- [287] D. Bursky, "Microcontrollers Span 8- and 16-bit Applications", *Electronic Design*, January 1992, pp.139-140.
- [288] M. J. Flynn et al, "And Now a Case for More Complex Instruction Sets", *Computer*, September 1987, pp.71-83.
- [289] A. Chang and M. F. Mergen, "801 Storage: Architecture and Programming", *ACM Transactions on Computer Systems*, Vol.6, No.1, February 1988, pp.28-50.
- [290] R. O. Simpson and P. D. Hester, "The IBM RT PC ROMP Processor and Memory Management Unit Architecture", *IBM Systems Journal*, Vol.26, No.4, April 1987, pp.346-360.
- [291] A. Lunde, "Empirical Evaluation of Some Features of Instruction Set Processor Architectures", *Communications of the ACM*, Vol.20, No.3, March 1977, pp.143-153.
- [292] M. E. Hopkins, "A Perspective on the 801/Reduced Instruction Set Computer", *IBM Systems Journal*, Vol.26, No.1, January 1987, pp.107-121.
- [293] P. W. Markstein, "Computation of Elementary Functions on the IBM RISC System/6000 Processor", *IBM Journal of Research and Development*, Vol.34, No.1, January 1990, pp.111-119.
- [294] J. Bond, "Boundary Scan Simplifies Board Test", *Test and Measurement World*, January 1991, pp.41-46.
- [295] Texas Instruments Inc., "New memory IC Stores Test History", *FYI*, Vol.9m Is.3, June 1992, p.2.
- [296] B. Fuller, "Performance Comes at a Price, Thanks to Frugal Developers", *Electronic Design*, June 1992.
- [297] D. Mann, "Speed System Operation by Matching CPU to Need", *Electronic Design*, November 1992, p.44-58.
- [298] Texas Instruments Inc., "Development/ Diagnostic Tool Gives Engineers Virtual Access", *FYI*, Vol.9, Is.5, September 1992, p.T2.

- [299] Texas Instruments Inc., "Open Formats Support IEEE 1149. 1 at Device/ Board or System Levels", FYI, September 1992.
- [300] Texas Instruments Inc., "Support for Embedded Scan-based Testing by Year's End", Vol.9, Is.5, FYI, September 1992, p.T3.
- [301] D. Bursky, "DSPs Expand Role as Cost Drops and Speed Increases", Electronic Design, October 1991, pp.53-81.
- [302] D. Wilson, "16-bit Microcontroller Packed with Peripherals", Computer Design, May 1992, p.111.
- [303] Motorola Inc., "New 68LC040 Lowers the Cost of High Performance", Semiconductor Data Update, November 1992.
- [304] Motorola Inc., "First 32-bit Microcontroller with Flash EEPROM Shines in High-performance Applications", Semiconductor Data Update, November 1992.
- [305] D. Bursky, "Flexible I/O Subsystem Gives MCU Many Resources", Electronic Design, October 1992, pp.53-58.
- [306] T. Williams, "Superscalar Sparc Chips Offer Performance Gains, Compatibility", Computer Design, July 1992, pp.32-36.
- [307] Intel Inc., "80960KB Hardware Designer's Reference Manual", Intel Corporation, 1988.
- [308] Motorola Inc., "MC68010/MC68012: 16-/32-bit Virtual Memory Microprocessors", Motorola, 1985.
- [309] Texas Instruments Inc., "TMS7000 Family Data Manual", Texas Instruments, 1986.
- [310] Motorola Inc., "MC88100 RISC Microprocessor User's Manual", Motorola, 1988.
- [311] Motorola Inc., "MC88200 Cache/Memory Management Unit User's Manual", Motorola, 1988.
- [312] BNR Inc., "BNR/CADENCE Course", BNR, 1990.
- [313] D. F. Barbe, "Very Large Scale Integration (VLSI): Fundamentals and Applications", Springer-Verlag, 1980.
- [314] Motorola Inc., "MC68332 User's Manual", Motorola, 1990.
- [315] D. L. Perry, "VHDL", McGraw-Hill Inc., 1991.
- [316] R. Wolfe and J. Graff, "Building Real-Time Distributed VXI Instrumentation Systems", Electronic Engineering, July 1991.
- [317] J. M. Graff, "Choosing the Right VXI Bus Controller", Test & Measurement World, May 1990.
- [318] W. Andrews, "PCs Take Many Forms in the Competition for Embedded Applications", Computer Design, April 1993.

APPENDICES

Appendix A Design Files

Appendix A1 - Design file types

The following table describes the various file formats for the files in this appendix.

FILE TYPE	FUNCTION
CFILE	For post-simulation viewing setup
EXEC	Operating system command file
FAULT	Readable fault simulation summary
FML	Low-level behaviour modeling language
FML2	High-level behaviour modeling language, may be compiled into data for simulation and into actual circuitry
FUNSIM	For automatic setup & exec of simulator
HOLIXZ	Describes timing & vectors for functional verification during simulation
LOGIC	Same as Macro, usually calls macro
MACRO	Specifies I/O and components in a block
SPEC	Specification text file

Table 21, Description of Design File Types

Appendix A2 - Design File Names

The following are the names of the computer files for the designed processor. The filenames' suffixes are as described in Appendix A1. Due to the fact Northern Telecom computer systems are required to utilize these files, the listings are not included in the thesis, but they are available upon request from the author.

**AIF CFILE
AIF FML2
AIF HOLIXZ
AIF MACRO**

**ALLM CFILE
ALLM FUNSIM
ALLM MACRO**

**ALU/ADDER MACRO
ALU/ADDER CFILE
ALU/ADDER HOLIXZ
ALU CFILE
ALU HOLIXZ
ALU LOGIC
ALU MACRO
ALU/AND MACRO
ALU/BTST MACRO
ALU/DP MACRO
ALU/FLAGS MACRO
ALU/MUX4 MACRO
ALU/MUX8 MACRO
ALU/MUXOUT MACRO
ALU/OR MACRO
ALU/REGA MACRO
ALU/SWAP MACRO**

**ALUS CFILE
ALUS FAULT
ALUS FML2
ALUS FUNSIM
ALUS HOLIXZ
ALUS MACRO**

**BRA CFILE
BRA FML2
BRA HOLIXZ
BRA MACRO**

CKC FML2
CKC MACRO

EPROM CFILE
EPROM FML2
EPROM FUNSIM
EPROM HOLIXZ
EPROM MACRO

MPC CFILE
MPC FML2 (MicroCode Listing)
MPC FUNSIM
MPC MACRO

RAM CFILE
RAM FML
RAM MACRO
RAM SPEC

RIF CFILE
RIF FML2
RIF FUNSIM
RIF HOLIXZ
RIF MACRO

RIM CFILE
RIM FUNSIM
RIM HOLIXZ
RIM MACRO

CAD4 EXEC

Appendix B - Calculations for Chip Area Estimates

Estimates of Chip Area	Using CMOS4 Technology
Area Gates is the area normalized to area of NAND2 cell	
Type Area is area in grids ² for all gates this type this module	
Area of NAND2 =	40 grids ²
Grid Size =	6.6 microns
Minimum transistor gate length=	1.5 microns
Minimum effective channel length =	1.2 microns

SUMMARY

ENTIRE CHIP AREA	12 mm ²
DOUBLED FOR ROUTING & I/O PADS	24 mm ²
EACH SIDE OF CHIP	4.9 m m

Module	SubModule	Gates	Count	Area Gates	Type Area [grids ²]	Module Area [μm ²] [mm ²]
AIF		MX2	16	2	1280	55757 0.056
ALU	SWAP	BUF	16	1	640	637718
	BTST	MUX2I	7	2	560	0.638
		INV	2	0.5	40	
		NAND2	1	1	40	
	ADDER	INV	18	0.5	360	
		MX2	16	2	1280	
		FADD	16	4	2560	
	FLAGS	MX2	1	2	80	
		OR8	2	4	320	
		OR2	1	1.5	60	
		NOR4	1	1.5	60	
		LN	2	2	160	
		LP	2	2	160	
	MUXOUT	MUX4	16	6.5	4160	
	REGA	DP	16	3.5	2240	
	AND	AND2	16	1.5	960	
	OR	OR2	16	1.5	960	
(continued...)						

(continued...)						
Module	SubModule	Gates	Count	Area Gates	Type Area [grids^2]	Module Area [μm^2] [mm^2]
BRA		AND2	4	1.5	240	148104
		AND3	4	2	320	0.148
		AND4	9	2	720	
		OR2	5	1.5	300	
		OR3	2	2	160	
		OR4	2	2	160	
		NOR2	1	1	40	
		INV	9	0.5	180	
		MX2	16	2	1280	
MPC		NOR4	13	1.5	780	2475079
		NOR3	17	2	1360	2.475
		NOR2	99	1	3960	
		NAND4	131	1.5	7860	
		NAND3	55	2	4400	
		NAND2	286	1	11440	
		INV	63	0.5	1260	
		INVB	7	1	280	
		INVP	3	1	120	
		INVD	2	6	480	
		AND4	9	2	720	
		AND3	3	2	240	
		AND2	13	1.5	780	
		AND0	16	3	1920	
		R				
		BUF	103	1	4120	
		BUFD	16	6	4320	
		INVC	3	2.5	300	
	for LSSD	DRSP	48	4.5	8640	
	for LSSD	MX2I	48	2	3840	
(...continued)						

(...continued)

Module	SubModule	Gates	Count	Area Gates	Type Area [grids^2]	Module Area [μm^2] [mm^2]
RIF	MX2		16	2	1280	62726
	AND2		2	1.5	120	0.063
	AND3		4	2		
	DRSP		4	4.5		
	INV		2	0.5	40	
RAM	Custom				195003	8557057 8.557
DVI	DEC38		1	19	760	229126
	BUFC		1	2	80	0.229
	MUX8		1	12	480	
	INV		2	0.5	40	
	OR3		1	2	80	
	AND2		1	1.5	60	
	S374		2	47	3760	

Appendix C - Original MC68000 application code

```

;-----
;
;BAR TESTER MC68000 ASSEMBLER HARDWARE CONTROL PROGRAM
;by M.Gazier, C.Y.Wu, P.Champagne.
;
;In this program Parallel port driver, OpticRAM driver, GPIB operation,
;EPROM to RAM Program Transfer, boot up/ power down sensor, timer
interface,
;& Analog Hardware Drivers are to be found. These include sequencer
;controller, ADC digitizers, DAC current source, multiplexers.
;Also, the timer is used to guarantee 0.1% duty cycle while testing
pulsed.
;
;                                     This is rev 4.4: 89 Mar 28
;
;-----
;ROM & RAM organization maps                                     (device #0 &
device #1)
ROMBASE EQU $0                      ;base address of ROM
RAMBASE EQU $100000                 ;base address of RAM, Software copied $0 to
$7FF
COMBUF EQU RAMBASE+$0800            ;base address for HPiB packet
communications
TABLE EQU RAMBASE+$1000            ;address of ramp table; should be even.
STACK EQU RAMBASE+$1FFF            ;top address of RAM for stack, grows
downwards
PWR EQU RAMBASE+$1FFE              ;power-up status: 0 = cold , 1 =
initialized
CW EQU RAMBASE+$1FFF              ;to remember if we are in CW or Pulsed mode
;-----
;HPiB interface label assignment
(device #2)
;
R3W EQU $200006                    ;auxiliary command register
R4W EQU $200008                    ;address register
R7W EQU $20000E                    ;data out register
R0R EQU $200000                    ;interrupt status register
R2R EQU $200004                    ;address status register
R3R EQU $200006                    ;auxiliary command register
R4R EQU $200008                    ;address switch register
R7R EQU $20000E                    ;data in register
;-----
;PI/T lable assignment
(device #3)
;
PIT EQU $300000                    ;base address for PI/T (offsets below are *
2 !)
PGCR EQU PIT+0                     ;port general control register
PADDR EQU PIT+4                    ;port A data direction register
PBDDR EQU PIT+6                    ;port B data direction register
PCDDR EQU PIT+8                    ;port C data direction register
PACR EQU PIT+12                    ;port A control register
PBCR EQU PIT+14                    ;port B control register
PADR EQU PIT+16                    ;port A data register

```

```

PBDR      EQU PIT+18          ;port B data register
PCDR      EQU PIT+24          ;port C data register
TCR       EQU PIT+32          ;Timer Control Register
TIVR      EQU PIT+34          ;Timer Interrupt Vector
CPRH      EQU PIT+38          ;Counter Preload Register High
CPRM      EQU PIT+40          ;Counter Preload Register Mid
CPRL      EQU PIT+42          ;Counter Preload Register Load
TSR       EQU PIT+52          ;Timer Status Register (contains ZDS flag)

```

```

;-----
;device #4 not defined at present, kept for EXPansion
(device #4)
;-----

```

```

;-----
;OpticRAM lable assignment
(device #5)

```

```

;
ROW       EQU $500000         ;starting ROW address
COL       EQU $520000         ;starting COLUMN address
B_READ    EQU $530000         ;starting READ address
;-----

```

```

;-----
;Analog Hardware Drivers label assignment                      (device #6 &
device #7)

```

```

XTEST     EQU $600000         ; Execute Test write address          (byte)
                                           ; The byte contains the device number
CWPUL     EQU $600001         ; Bit 0 is 0 for CW 1 for pulsed    (byte)
IWR       EQU $700000         ; ADC current source write address (word)
IRD       EQU $700000         ; DAC #0 read address      (I )    (word)
VRD       EQU $710000         ; DAC #1 read address      (V )    (word)
L1RD      EQU $720000         ; DAC #2 read address      (L1)    (word)
L2RD      EQU $730000         ; DAC #3 read address      (L2)    (word)
;-----

```

```

;-----
;Command Character Lables - These are the bytes sent to branch to
subprograms

```

```

;
AMASK     EQU $41             ;mask of char 'A'
BMASK     EQU $42             ;mask of char 'B'
CMASK     EQU $43             ;mask of char 'C'
GMASK     EQU $47             ;mask of char 'G'
HMASK     EQU $48             ;mask of char 'H'
IMASK     EQU $49             ;mask of char 'I'
MMASK     EQU $4D             ;mask of char 'M'
OMASK     EQU $4F             ;mask of char 'O'
PMASK     EQU $50             ;mask of char 'P'
QMASK     EQU $51             ;mask of char 'Q'
RMASK     EQU $52             ;mask of char 'R'
SMASK     EQU $53             ;mask of char 'S'
TMASK     EQU $54             ;mask of char 'T'
UMASK     EQU $55             ;mask of char 'U'
VMASK     EQU $56             ;mask of char 'V'
WMASK     EQU $57             ;mask of char 'W'
XMASK     EQU $58             ;mask of char 'X'
;-----

```

```

;-----
;program constants not stored in memory but just used in assembler prog.
; HPIB Interface

```

```

EOIMASK EQU $20          ;mask for End Of Identify bit (EOI)
ASSRESET EQU $80         ;mask value to assert reset
NEGRESET EQU $00         ;mask value to negate reset
; Parallel Port Interface
MODE EQU $0              ;pit mode
SUBMODE EQU $80          ;pit submode
OUTBIT EQU $FF           ;set port to 8-bit output
INBIT EQU $0             ;set port to 8-bit input
INITPA EQU $3            ;port A initial value (inker out, vacuum
ports
                           ;off, arm up)
INITPB EQU $0            ;port B initial value (all vacuum off)
INITPC EQU $0            ;port C initial value (all vacuum off)
; Timer Interface
TENB EQU $1              ;enable for timer in port TCR
TDIS EQU $0              ;disable for timer in port TCR
TINTVEC EQU $0           ;timer interrupt vector init (not used)
; OpticRAM Interface
COLMAX EQU $200          ;maximum COLUMN number=512
ROWMAX EQU $80           ;maximum ROW number=128
; ROM to RAM subroutine
ROMSIZE EQU $B00         ;# words for ROMRAM to transfer
;-----
;Interrupt Vector List - Reset Vector and Program Vector for power
up/reset
                ORG 0
                DC.L STACK          ; Stack Vector
                DC.L MAIN           ; Program Vector
;-----
;-----
;Table of constants which are stored as constants in memory
                ORG $100
REV:            DC.B 42             ;revision number times 10 (ex:4.2 is 42)
YEAR:          DC.B 89             ;date this EPROM was programmed and burned
MONTH:         DC.B 03             ;this acts as a signature so we know which
DAY:           DC.B 23             ;revision we have, can be 'Queried Version'
IOFF:          DC.W $FFFF          ;value to turn current source off after
ramp
BYTEM:         DC.L $000000FF      ;mask out just the lower byte
MSKBIT:        DC.L $00000001      ;mask out just the most significant bit
RES:           DC.L $00008000      ;rescale for transfer to HP9000 as INTEGER
LCH:           DC.B $00            ;timer preload count values for testing
LCM:           DC.B $06            ;together= 1563 cycles = 5msec
LCL:           DC.B $1B            ;for the 0.1% Duty Cycle guarantee
                           ;actual system is faster really
ICH:           DC.B $00            ;timer preload count values for inking
ICM:           DC.B $3D            ; which = 50msec
ICL:           DC.B $09
;Tables of data for re-mapping barholders' port # to PIT port #
;This is necessary because PIT ports are "randomly" used for functions
;Is organized as barport#0[PIT A,PIT B,PIT C],#1[A,B,C],...
;The actual bit number is pointed to. ex: bit #2 would have a 2^2=4
below.
PTRA: DC.B 4,0,0,8,0,0,16,0,0,32,0,0,64,0,0,128,0,0,0,64,0,0,128,0
;vacuum A
PTRB: DC.B 0,2,0,0,1,0,0,4,0,0,8,0,0,16,0,0,32,0,0,0,1,0,0,2
;vacuum B

```



```

-----
;
-----
                ORG $200          ; starting location of program
-----
; ISET :-input      : ---
ISET
;
;   -output         : ---
;   -subroutines    : GETCHR
;   -reg. affected: D0,D1
;   -description    : two bytes are received over the HPiB: MSB, LSB
;                   : these set a DC current on the current source card
ISET:    BSR GETCHR
          MOVE.L D1,D0          ;inputs MSB of the set-current word
          LSL.L #8,D0
          BSR GETCHR          ;inputs LSB of the set-current word
          OR.L D1,D0
          MOVE.W D0,IWR        ;output set-current to card
          RTS
-----
;
-----
; RAMPT:-input      : ---
RAMPT
;
;   -output         : ---
;   -subroutines    : GETCHR
;   -reg. affected: A0,A1,A2,A3,A4,A5,D0,D1,D2,D3,D4,D5,D6
;   -description    : Using the ramp table we test lasers #1 to
end_laser,
;                   : ramping up using 16-bit values in the table and
send
;                   : the values back. #of steps is in table+0.
;                   : The data is sent back in "packets". One packet
;                   : is sent for every step in the current ramp. This
;                   : yields faster communication. For 10 lasers
testing
;                   : one packet is 10 lasers*8 bytes/lasers=80 bytes
;                   : The channels are : 0: current
(I)
;                   :
;                   : 1: voltage
(V)
;                   :
;                   : 2: front facet photodetector
(PD1)
;                   :
;                   : 3: back facet photodetector
(PD2)
;
;                   : At the begin of each current step the timer is
;                   : triggered. After communication is done, the timer
;                   : is checked and the system waits for time to
elapse.
;                   :
;                   : This ensures 0.1% Duty Cycle during pulsed
operation.
;                   :
;                   : This is not done when CW testing is performed.
RAMPT:    BSR GETCHR          ;inputs value of End_laser in D1
          MOVEA.L #TABLE,A0   ;pointer to ramp table
          MOVEA.L #IWR,A1     ;pointer to current set port
          MOVEA.L #XTEST,A2   ;ponter to test laser port
          MOVEA.L #R0R,A3     ;location of HPiB ready bit
          MOVEA.L #R7W,A4     ;HPiB output port register location
          MOVE.W (A0)+,D0     ;# of words in ramp table

```

```

format      MOVE.L RES,D5          ;offset for INTEGER communications
high        MOVE.B LCH,CPRH        ;initialize counter preload register
            MOVE.B LCM,CPRM        ;initialize counter preload register mid
            MOVE.B LCL,CPRL        ;initialize counter preload register low

NEXTCUR:    MOVE.W (A0)+,(A1)      ;set output current to next value in
table
            MOVEA.L #COMBUF,A5    ;pointer in HPIB buffer for packet
sending
            BTST.B #0,CW          ;Set up Duty Cycle control in Pulsed
only
            BEQ ISKIP
            MOVE.B #TENB,TCR      ;Start timer (enable) for Duty Cycle
control
ISKIP:      CLR.L D3              ;start test at laser #D3=0

NEXTLAS:    ADDQ.W #1,D3          ;point to next laser (#1 to End_laser)
[TEST]
            MOVE.B D3,(A2)        ;pulse and test laser #D3
            MOVE.W IRD,(A5)       ;read current channel. store in
combuffer
            ADD.W D5,(A5)+
            MOVE.W VRD,(A5)       ;read voltage channel
            ADD.W D5,(A5)+
            MOVE.W L1RD,(A5)      ;read front facet channel
            ADD.W D5,(A5)+
            MOVE.W L2RD,(A5)      ;read back facet channel
            ADD.W D5,(A5)+
            CMP.W D3,D1           ;does D1-D3
            BGT NEXTLAS          ;next laser at this current step, if any

[SEND]
            MOVEA.L #COMBUF,A5    ;base address of communications buffer
            CLR.L D3              ;send back a packet for this current
step
BL:         ADDQ.W #1,D3          ;sending back laser #D3 data
WL1:        BTST.B #06,(A3)       ;wait for HPIB clear to send
            BEQ WL1
            MOVE.B (A5)+,(A4)     ;send over bus this laser's byte #1
WL2:        BTST.B #06,(A3)       ;wait for HPIB clear to send
            BEQ WL2
            MOVE.B (A5)+,(A4)     ;send over bus this laser's byte #2
WL3:        BTST.B #06,(A3)       ;wait for HPIB clear to send
            BEQ WL3
            MOVE.B (A5)+,(A4)     ;send over bus this laser's byte #3
WL4:        BTST.B #06,(A3)       ;wait for HPIB clear to send
            BEQ WL4
            MOVE.B (A5)+,(A4)     ;send over bus this laser's byte #4
WL5:        BTST.B #06,(A3)       ;wait for HPIB clear to send
            BEQ WL5
            MOVE.B (A5)+,(A4)     ;send over bus this laser's byte #5
WL6:        BTST.B #06,(A3)       ;wait for HPIB clear to send
            BEQ WL6
            MOVE.B (A5)+,(A4)     ;send over bus this laser's byte #6
WL7:        BTST.B #06,(A3)       ;wait for HPIB clear to send
            BEQ WL7

```

```

        MOVE.B (A5)+, (A4)      ;send over bus this laser's byte #7
WL8:    BTST.B #06, (A3)      ;wait for HPIB clear to send
        BEQ WL8
        MOVE.B (A5)+, (A4)      ;send over bus this laser's byte #8
        CMP.W D3, D1           ;does D1-D3
        BGT BL                 ;Send for next laser

        BTST.B #0, CW          ;check timer for 0.1% duty cycle
        BEQ SWAIT              ;done in Pulsed mode only
WAITF:  BTST.B #0, TSR         ;check if 5msec is up
        BEQ WAITF
        MOVE.B #TDIS, TCR      ;disable timer and prepare for next time

SWAIT:  SUBQ.W #1, D0           ;next current in ramp table
        BNE NEXTCUR
        MOVE.W IOFF, (A1)      ;turn off current source after test
        RTS

;-----
;TABLEIN:-input      : ---
TABLEIN
;      -output       : ---
;      -subroutines   : GETCHR
;      -reg. affected: D0, D1, A0
;      -description   : we upload and prepare the ramp table for the
;                      automatic ramping subroutine
TABLEIN: BSR GETCHR
        MOVE.L D1, D0          ;inputs number of data points, up to 256
here
        MOVEA.L #TABLE, A0     ;pointer in table
        MOVE.W D0, (A0)+       ;store #words into table+0
        LSL.W #1, D0           ;convert #words to #bytes
RALP:   BSR GETCHR             ;download the words as bytes
        MOVE.B D1, (A0)+
        SUBQ.W #1, D0
        BNE RALP
        RTS

;-----
;ROCH :-input        : ---
ROCH
;      -output       : ---
;      -subroutine    : GETCHR, SENDCHR
;      -reg. affected: D0, D1, D2, A0
;      -description   : the four ADC channels are read from the ADC cards
and
;                      then sent back over the HPIB:
MSB0, LSB0, ..., MSB3, LSB3
;                      where the channels are : 0: current
;                      1: voltage
;                      2: front facet
photodetector
;                      3: back facet
photodetector
ROCH:   MOVEA.L #R0R, A0
        MOVE.B IRD, D2         ;read current channel MSB
RS1:    BTST.B #06, (A0)       ;waits for BO (byte out) to be asserted
        BEQ RS1               ;and sends D2 over HPIB

```

```

        MOVE.B D2,R7W           ;output to HPID data out register
        MOVE.B IRD+1,D2         ;read current channel LSB
RS2:    BTST.B #06,(A0)
        BEQ RS2
        MOVE.B D2,R7W
        MOVE.B VRD,D2           ;read voltage channel MSB
RS3:    BTST.B #06,(A0)
        BEQ RS3
        MOVE.B D2,R7W
        MOVE.B VRD+1,D2         ;read voltage channel LSB
RS4:    BTST.B #06,(A0)
        BEQ RS4
        MOVE.B D2,R7W
        MOVE.B L1RD,D2          ;read front facet channel MSB
RS5:    BTST.B #06,(A0)
        BEQ RS5
        MOVE.B D2,R7W
        MOVE.B L1RD+1,D2        ;read front facet channel LSB
RS6:    BTST.B #06,(A0)
        BEQ RS6
        MOVE.B D2,R7W
        MOVE.B L2RD,D2          ;read back facet channel MSB
RS7:    BTST.B #06,(A0)
        BEQ RS7
        MOVE.B D2,R7W
        MOVE.B L2RD+1,D2        ;read back facet channel LSB
RS8:    BTST.B #06,(A0)
        BEQ RS8
        MOVE.B D2,R7W
        RTS

;-----
;EXEC :-input      : ---
EXEC
;      -output      : ---
;      -subroutine   : GETCHR
;      -reg. affected: D1
;      -description  : the number of the laser to test is input from the
;                      HPID and then a test sequence is done
EXEC:    BSR GETCHR           ; input the number of the laser to test
        MOVE.B D1,XTEST      ; go test it
        RTS

;-----
;CWORPUL: -input    : ---
CWORPUL
;      -output      : ---
;      -subroutine   : GETCHR
;      -reg. affected: D1
;      -description  : a number is input from the HPID port to select
;                      the test mode: CW (0) or Pulsed (1)
CWORPUL: BSR GETCHR           ; input the choice for DC or PULSED
        MOVE.B D1,CWPUL      ; go set it
        MOVE.B D1,CW         ; store into CW for later reference
        RTS

;-----
;ROM TO ROM :-input : ---

```

```

ROMRAM
;TRANSFER      :-output      : ---
;
;               -subroutines : ---
;
;               -reg. affected: A0,A1,D0,D1
;
;               -description : transfer the program to RAM from the EPROM
ROMRAM: MOVEA.L #ROMBASE,A0
        MOVEA.L #RAMBASE,A1
        MOVE.L  #ROMSIZE,D0      ;counter for number of words remaining
LOOP:   MOVE.W  (A0)+,D1
        MOVE.W  D1,(A1)+
        SUBQ.L  #1,D0
        BNE     LOOP
        RTS

;-----
;HPIBINIT:-input      : ---
HPIBINIT
;
;               -output      : ---
;
;               -subroutines : ---
;
;               -reg. affected: D0
;
;               -description : initialize the HPIB port
HPIBINIT:MOVE.W #ASSRESET,D0 ;asserts reset bit in reg. R3W
        MOVE.B D0,R3W
        MOVE.W #NEGRESET,D0 ;negates reset bit in reg. R3W
        MOVE.B D0,R3W
        MOVE.B R4R,D0      ;read HPIB address from switch on board
(via ASE)
        MOVE.B D0,R4W      ;writes address to address register in HPIB
chip
        RTS

;-----
;TIMER: -input      : ---
TIMER
;
;               -output      : ---
;
;               -subroutines : GETCHR
;
;               -reg. affected: ---
;
;               -description : input three bytes (High,Mid,Low) and delay that
time
;                               on the built in timer (3.2usec/count @ 10MHz
clock)
TIMER:   BSR GETCHR
        MOVE.B D1,CPRH      ;initialize counter preload register
high
        BSR GETCHR
        MOVE.B D1,CPRM      ;initialize counter preload register mid
        BSR GETCHR
        MOVE.B D1,CPRL      ;initialize counter preload register low
        MOVE.B #TENB,TCR    ;Start timer (enable) for Duty Cycle
control
WAITTR:  BTST.B #0,TSR      ;check if time is up
        BEQ WAITTR
        MOVE.B #TDIS,TCR   ;disable timer and prepare for next time
        BSR SENDCHR
        RTS

;-----
;GETCHR:-input      : ---

```

```

GETCHR
;      -output      : D1
;      -subroutines : ---
;      -reg. affected: D1
;      -description : read a character from the HPIB port
GETCHR: BTST.B #0,R0R          ;verify that bit BI (byte in) is
asserted
        BEQ GETCHR
        MOVE.B R7R,D1          ;read data-in register
        AND.L BYTEM,D1
        RTS

;-----
;-----
;VERIFCHR:-input      : D1
VERIFCHR
;      -output      : ---
;      -subroutines : RAMTST,PITTST,SENDCHR
;      -reg. affected: D2,D6
;      -description : compare a character to commands and execute
command
VERIFCHR: CMPI.B #GMASK,D1      ; 'G': test laser using ramp table
        BEQ RAMPT
        CMPI.B #HMASK,D1      ; 'H': upload ramp table
        BEQ TABLEIN
        CMPI.B #WMASK,D1      ; 'W': write to parallel out
subcommand
        BEQ PITW
        CMPI.B #OMASK,D1      ; 'O': grab an image from camera-
OpticRAM
        BEQ OP_RAM
        CMPI.B #IMASK,D1      ; 'I': set output current on DAC
        BEQ ISET
        CMPI.B #RMASK,D1      ; 'R': read 4 ADCs
        BEQ ROCH
        CMPI.B #TMASK,D1      ; 'T': exec laser test on one laser
        BEQ EXEC
        CMPI.B #CMASK,D1      ; 'C': select CW or PULsed mode
        BEQ CWORPUL
        CMPI.B #QMASK,D1      ; 'Q': query subcommand
        BEQ QSUB
        RTS                  ; invalid command, ignore
QSUB:   BSR GETCHR            ; find out what is queried (aux
function)
        CMPI.B #CMASK,D1      ; -'C' is query DC/Pulsed
        BNE NOC
        MOVE.B CW,D2
        BRA SENDCHR           ; send data back and return
NOC:    CMPI.B #PMASK,D1      ; -'P' is query power up status
        BNE NOP
        MOVE.B PWR,D2
        BRA SENDCHR           ; send data back and return
NOP:    CMPI.B #RMASK,D1      ; -'R' is read back 3 Parallel Ports
        BEQ PITR
        CMPI.B #WMASK,D1      ; -'W' is do a delay using Timer
(Wait)
        BEQ TIMER
        CMPI.B #SMASK,D1      ; -'S' is to set power up flag to
'warm'

```

```

        BNE QIG
        MOVE.B #1,PWR                ;set power status = initialized
        BRA QIG2
QIG:    CMPI.B #VMASK,D1              ; -'V' is to Query Version (date &
rev#)                                     ;
        BNE QIG2                    ;invalid query then ignore
        MOVE.B REV,D2                ;send REV#
        BSR SENDCHR
        MOVE.B YEAR,D2               ;send YEAR
        BSR SENDCHR
        MOVE.B MONTH,D2              ;send MONTH
        BSR SENDCHR
        MOVE.B DAY,D2                ;send DAY
        BRA SENDCHR
QIG2:   RTS
;-----
;SENDCHR: -input          :D2
SENDCHR
;      -output           : ---
;      -subroutines      : ---
;      -reg. affected: D3
;      -description      : sends data over HPID, with EOI set each time
SENDCHR: BTST.B #06,R0R              ;waits for BO (byte out) to be
asserted                               ;
        BEQ SENDCHR                 ,and sends D2 over HPID, with EOI
set.
        MOVEQ #EOIMASK,D3           ;destroys previous register R3W
        MOVE.B D3,R3W
        MOVE.B D2,R7W               ;output to HPID data out register
        RTS
;-----
;PITINIT: -input          : -
PITINIT
;      -output           : -
;      -subroutines      : -
;      -reg. affected: -
;      -description      : initialize ports A,C as output, port B as
input
PITINIT: MOVE.B #OUTBIT,PADDR        ;set port A to 8-bit output
        MOVE.B #OUTBIT,PBDDR        ;set port B to 8-bit output
        MOVE.B #OUTBIT,PCDDR        ;set port C to 8-bit output
        MOVE.B #SUBMODE,PACR        ;set port A to submode 1X
        MOVE.B #SUBMODE,PBCR        ;set port B to submode 1X
        MOVE.B #MODE,PGCR           ;enable port A and B, mode 0
        MOVE.B #TINTVEC,TIVR        ;set timer's interrupt vector to point
to 0
        MOVE.B #TDIS,TCR            ;initialize to: port C is I/O port,
;no zero rollover, use divide by 32
prescaler
;with internal clock (10MHz=3.2usec per
;cycle), timer off.
        MOVE.B #INITPA,PADR         ;initialize port A outputs to boot
value
        MOVE.B #INITPB,PBDR         ;initialize port B outputs to boot
value
        MOVE.B #INITPC,PCDR         ;initialize port C outputs to boot

```

value

RTS

```

;-----
;PITR: -input      : -
PITR
;      -output      : -
;      -subroutines : SENDCHR
;      -reg. affected: D2
;      -description : PIT port B is read and the # sent over HPIB
PITR:  MOVE.B PADR,D2          ;read & send pit port A over HPIB
      BSR SENDCHR
      MOVE.B PBDR,D2          ;read & send pit port B over HPIB
      BSR SENDCHR
      MOVE.B PCDR,D2          ;read & send pit port C over HPIB
      BSR SENDCHR
      RTS
;-----

```

```

;-----
;PITW: -input      : -
PITW
;      -output      : -
;      -subroutines : GETCHR
;      -reg. affected: D1,A0
;      -description : "Intelligent" control of Parallel Ports
;                      Available Commands (over HPIB) are:
;                      1. A#V   2. B#V   3. I   4. RS
;                      ,where 'A' & 'B' indicate port A or port B
;                      ,# is the barholder port # (range=0 to 7)
;                      , 'I' is for ink a dot (wait 0.1s,in,wait
0.1s,out)
;                      , 'R' is for arm control
;                      ,V=0 for vacuum off, 1 for vacuum on, 2 for
toggle
;                      ,S='U' for arm Up, 'D' for arm DOWN
PITW:  BSR GETCHR              ;get a command from controller
      CMPI.B #IMASK,D1        ;ink a dot command
      BEQ INKIT
      CMPI.B #RMASK,D1        ;move arm command
      BEQ ARMMOV
      CMPI.B #AMASK,D1        ;vacuum port A command
      BEQ VPA
      CMPI.B #BMASK,D1        ;vacuum port B command
      BEQ VPB
      RTS
; ink a dot
INKIT: BSR WAIT01              ; ink a dot. first wait 0.1s
      EORI.B #$2,PADR         ; inker is A1, push it down
      BSR WAIT01              ; wait another 0.1s
      EORI.B #$2,PADR         ; pull inker out
      RTS
; delay of 0.1 sec using timer for inker
WAIT01: MOVE.B ICH,CPRH        ;initialize counter preload register
high
      MOVE.B ICM,CPRM         ;initialize counter preload register
mid
      MOVE.B ICL,CPRL         ;initialize counter preload register
low

```



```

        BNE QIG
        MOVE.B #1,PWR                ;set power status = initialized
        BRA QIG2
QIG:    CMPI.B #VMASK,D1              ; -'V' is to Query Version (date &
rev#)                                     ;
        BNE QIG2                    ;invalid query then ignore
        MOVE.B REV,D2                ;send REV#
        BSR SENDCHR
        MOVE.B YEAR,D2               ;send YEAR
        BSR SENDCHR
        MOVE.B MONTH,D2              ;send MONTH
        BSR SENDCHR
        MOVE.B DAY,D2                ;send DAY
        BRA SENDCHR
QIG2:   RTS
;-----
;SENDCHR: -input      :D2
SENDCHR
;      -output      : ---
;      -subroutines  : ---
;      -reg. affected: D3
;      -description  : sends data over HPIB, with EOI set each time
SENDCHR: BTST.B #06,R0R              ;waits for BO (byte out) to be
asserted
        BEQ SENDCHR                 ;and sends D2 over HPIB, with EOI
set.
        MOVEQ #EOIMASK,D3           ;destroys previous register R3W
        MOVE.B D3,R3W
        MOVE.B D2,R7W               ;output to HPIB data out register
        RTS
;-----
;PITINIT: -input      : -
PITINIT
;      -output      : -
;      -subroutines  : -
;      -reg. affected: -
;      -description  : initialize ports A,C as output, port B as
input
PITINIT: MOVE.B #OUTBIT,PADDR        ;set port A to 8-bit output
        MOVE.B #OUTBIT,PBDDR        ;set port B to 8-bit output
        MOVE.B #OUTBIT,PCDDR        ;set port C to 8-bit output
        MOVE.B #SUBMODE,PACR        ;set port A to submode 1X
        MOVE.B #SUBMODE,PBCR        ;set port B to mode 1X
        MOVE.B #MODE,PGCR           ;enable port A and B, mode 0
        MOVE.B #TINTVEC,TIVR        ;set timer's interrupt vector to point
to 0
        MOVE.B #TDIS,TCR            ;initialize to: port C is I/O port,
prescaler                               ;no zero rollover, use divide by 32
;with internal clock (10MHz=3.2usec per
;cycle), timer off.
        MOVE.B #INITPA,PADR         ;initialize port A outputs to boot
value
        MOVE.B #INITPB,PBDR         ;initialize port B outputs to boot
value
        MOVE.B #INITPC,PCDR         ;initialize port C outputs to boot

```

```

value
      RTS
;-----
;-----
;PITR: -input      : -
PITR
;      -output      : -
;      -subroutines : SENDCHR
;      -reg. affected: D2
;      -description : PIT port B is read and the # sent over HPiB
PITR:  MOVE.B PADR,D2          ;read & send pit port A over HPiB
      BSR SENDCHR
      MOVE.B PBDR,D2          ;read & send pit port B over HPiB
      BSR SENDCHR
      MOVE.B PCDR,D2          ;read & send pit port C over HPiB
      BSR SENDCHR
      RTS
;-----
;-----
;PITW: -input      : -
PITW
;      -output      : -
;      -subroutines : GETCHR
;      -reg. affected: D1,A0
;      -description : "Intelligent" control of Parallel Ports
;                      Available Commands (over HPiB) are:
;                      1. A#V   2. B#V   3. I   4. RS
;                      ,where 'A' & 'B' indicate port A or port B
;                      ,# is the barholder port # (range=0 to 7)
;                      , 'I' is for ink a dot (wait 0.1s,in,wait
0.1s,out)
;                      , 'R' is for arm control
;                      ,V=0 for vacuum off, 1 for vacuum on, 2 for
toggle
;                      ,S='U' for arm Up, 'D' for arm DOWN
PITW:  BSR GETCHR              ;get a command from controller
      CMPI.B #IMASK,D1        ;ink a dot command
      BEQ INKIT
      CMPI.B #RMASK,D1        ;move arm command
      BEQ ARMMOV
      CMPI.B #AMASK,D1        ;vacuum port A command
      BEQ VPA
      CMPI.B #BMASK,D1        ;vacuum port B command
      BEQ VPB
      RTS
; ink a dot
INKIT: BSR WAIT01              ; ink a dot. first wait 0.1s
      EORI.B #$2,PADR         ; inker is A1, push it down
      BSR WAIT01              ; wait another 0.1s
      EORI.B #$2,PADR         ; pull inker out
      RTS
; delay of 0.1 sec using timer for inker
WAIT01: MOVE.B ICH,CPRH        ;initialize counter preload register
high
      MOVE.B ICM,CPRM          ;initialize counter preload register
mid
      MOVE.B ICL,CPRL          ;initialize counter preload register
low

```

```

control      MOVE.B #TENB,TCR          ;Start timer (enable) for Duty Cycle
WAITA:      BTST.B #0,TSR              ;check if 0.1 sec up
            BEQ WAITA
            MOVE.B #TDIS,TCR          ;disable timer and prepare for next
time
            RTS
; move the arm up or down
ARMMOV:     BSR GETCHR                  ; obtain if move up or move down
            CMPI.B #UMASK,D1           ; 'U'=arm up, 'D'=arm down
            BEQ ARMUP
            ANDI.B #$FE,PADR          ; ARM down is 0 in bit A0
            RTS
ARMUP:      ORI.B #$1,PADR              ; ARM up is 1 in bit A0
            RTS
; turn a vacuum port on or off
VPA:        BSR GETCHR                  ; set vacuum port A. get port # in D1
            MOVEA.L #(RAMBASE+PTRA),A0 ; address of table barholder A
            BRA LPO
VPB:        BSR GETCHR                  ; set vacuum port B. get port # in D1
            MOVEA.L #(RAMBASE+PTRB),A0 ; address of table barholder B
LPO:        CMPI.B #0,D1                ; are we done adding offsets
            BEQ VPC                    ; find offset for data in table
            ADDA.W #3,A0                ; each group of A,B,C is 3 long
            SUBQ.B #1,D1
            BRA LPO
VPC:        BSR GETCHR                  ; get On(D1=1) or Off(D1=0) or
Toggle(D1=2)
            BNE DOTHEON
            MOVE.B (A0)+,D1             ; if Off then must flip bits for AND
            EORI.B #$FF,D1
            AND.B D1,PADR               ; do port A
            MOVE.B (A0)+,D1             ; if Off then must flip bits for AND
            EORI.B #$FF,D1
            AND.B D1,PBDR               ; do port B
            MOVE.B (A0)+,D1             ; if Off then must flip bits for AND
            EORI.B #$FF,D1
            AND.B D1,PCDR               ; do port C
            RTS
DOTHEON:    CMPI.B #2,D1                ; for toggle case
            BEQ TOGGLEV
            MOVE.B (A0)+,D1             ; if On the just set the bit mapped
using OR
            OR.B D1,PADR                ; do port A
            MOVE.B (A0)+,D1
            OR.B D1,PBDR                ; do port B
            MOVE.B (A0)+,D1
            OR.B D1,PCDR                ; do port C
            RTS
TOGGLEV:    MOVE.B (A0)+,D1             ; if Toggle then flip (XOR) the good
bit
            EOR.B D1,PADR               ; do port A
            MOVE.B (A0)+,D1
            EOR.B D1,PBDR               ; do port B
            MOVE.B (A0)+,D1
            EOR.B D1,PCDR               ; do port C
            RTS
;-----

```

```

-----
;OpticRAM:-input      : -
OPTICRAM
;      -output      : -
;      -subroutine   : GETCHR
;      -reg. affected: D0,D1,D3,D4,D5,D7,A0,A1,A2,A3,A4,A5,A6,A7
;      -description  : in this subroutine, the exposure time is input
from
;
;      the user (HPIB), a Page Mode Write is
performed
;
;      in which the first array (128x512) of the
OpticRAM
;
;      is initialized to 1's; then the image is
exposed
;
;      on the array for about 500ms; then a Page Mode
Read
;
;      is performed, in which either a 1 or a 0 is
read
;
;      from the OpticRAM & transfered over the HPIB
;
;      every 16 pixels; also, a BURST refresh in
which all
;
;      rows are accessed once (column 0) to refresh
the
;
;      whole row is done every 10 rows read in.
;
OP_RAM:  BSR GETCHR          ;get exposure time,MSB first
         MOVE.L D1,D5
         LSL.L #8,D5
         BSR GETCHR          ;input exposure LSB
         OR.L D1,D5          ;D5 as the exposure time counter

;page mode WRITE cycle
WRITEINI:CLR.L D0            ;clear max. row counter
          CLR.L D1            ;clear max. column counter
          MOVE.W #ROWMAX,D0    ;load max. row number(128) into D0
          MOVEA.L #ROW,A0      ;load starting row addr into A0
          MOVEA.L #COL,A1      ;load starting col addr into A1
          MOVEA.L #B_READ,A6   ;load starting read bit into A6
          MOVE.B #1,(A0)       ;make RAS active high
          MOVE.B #1,(A1)       ;make CAS active high
          MOVE.W #$11,(A0)     ;to make WE active low
WRITE:    MOVE.B #0,(A0)       ;to make RAS active low
          MOVE.W #COLMAX,D1    ;load max col number(512) into D1
DUMP:     MOVE.B #0,(A1)       ;to make CAS active low
          MOVE.B #1,(A6)       ;initialize array to 1's
          MOVE.B #1,(A1)       ;to make CAS active high
          ADDA.L #2,A1          ;increment COL address
          ADDA.L #2,A6
          SUBQ.W #1,D1          ;decrease COL counter by 1
          BNE DUMP
          MOVE.B #1,(A0)       ;to make RAS active high
          ADDA.L #2,A0          ;increment ROW address
          MOVEA.L #COL,A1      ;init. to COL 0 for next ROW
          MOVEA.L #B_READ,A6
          SUBQ.W #1,D0          ;decrease ROW counter by 1
          BNE WRITE

;expose the image onto the array

```

```

SOAK:    MOVE.L D5,D1           ;approximately 50 usec per loop
        MOVE.L #$A,D0         ;save the exposure time
        DIVS #$A,D0           ;just want to kill some time
        MULU #$A,D0           ;kill time
        DIVS #$A,D0           ;while expose an image
        MULU #$A,D0
        SUB.L #1,D1
        BNE SOAK              ;decrease exposure counter by 1
                                   ;if < 500ms, go to SOAK

INIT:    CLR.L D0              ;page mode READ cycle
        CLR.L D1              ;clear max. row counter
        CLR.L D3              ;clear max. column counter
        CLR.L D3              ;clear 10 row counter (every 10 rows do
BURST)    CLR.L D4              ;clear shift register counter
        CLR.L D5              ;temporary storage for data
        CLR.L D6              ;temporary storage
        CLR.L D7

        MOVEA.L #ROW,A2       ;for every 10 rows do a BURST Refresh
        MOVEA.L #COL,A3
        MOVEA.L #B_READ,A4
        MOVE.W #ROWMAX,D2
        MOVE.W #$10,(A2)

FRESH1:  MOVE.B #0,(A2)        ;set RAS low
        MOVE.B #0,(A3)
        MOVE.B (A4),(A5)      ;refresh entire array once
        MOVE.B #1,(A2)        ;set RAS back to high
        MOVE.B #1,(A3)
        ADDA.L #2,A2          ;increment row address
        ADDA.L #2,A3
        ADDA.L #2,A4
        SUBQ.W #1,D2          ;decrease counter by 1
        BNE FRESH1

        MOVE.B #10,D3         ;D3 as a 10 row burst one time counter
        MOVE.B #16,D4         ;send 8 pixels at a time
        MOVE.W #ROWMAX,D0
        MOVEA.L #3_READ,A6
        MOVEA.L #ROW,A0
        MOVEA.L #COL,A1
        MOVE.B #1,(A0)        ;make RAS active high, write an 1 to D8
        MOVE.B #1,(A1)        ;make CAS active high, write an 0 to D8
        MOVE.W #$10,(A0)      ;to make WE active high (0 to D8,1 to
D0)
READ:    MOVE.B #0,(A0)        ;make RAS active low
        MOVE.W #COLMAX,D1

DATA:    MOVE.B #0,(A1)        ;make CAS active low
        MOVE.B (A6),D5        ;read from opticRAM to temp storage
        AND.L MSKBIT,D5       ;mask out just the last bit
        BEQ PASS              ;if the last bit is 0 then skip
        ADDQ.W #1,D7          ;the bit is 1, add 1 to counter
PASS:    ADD.W D5,D6           ;save data in D6
        SUBQ.B #1,D4          ;dec. counter
        BNE CONT              ;if < 16 bits, continue reading

SENDIMG: MOVE.W D6,D2          ;if = 16 bits send this portion

```

```

        LSR.W #8,D2
SBY1:   BTST.B #06,R0R
        BEQ SBY1
        MOVE.B D2,R7W
SBY2:   BTST.B #06,R0R
        BEQ SBY2
        MOVE.B D6,R7W

        MOVE.B #16,D4           ;reload the 8 bit counter
        CLR.L D6
CONT:   LSL.L #1,D6             ;shift 1 bit to the left
        MOVE.B #1,(A1)         ;set CAS active high
        ADDA.L #2,A1           ;increment column address
        ADDA.L #2,A6           ;increment Read Bit address
        SUBQ.W #1,D1           ;decrease COL counter by 1
        BNE DATA
        MOVE.B #1,(A0)         ;set RAS high
        ADDA.L #2,A0           ;increment row address
        MOVEA.L #COL,A1        ;init. to COL 0 for next ROW
        MOVEA.L #B_READ,A6     ;initialize for next ROW
        SUBQ.B #1,D3           ;decrease 10 row counter
        BNE SKIP              ;if < 10, continue reading

        MOVEA.L #ROW,A2        ;for every 10 rows do a BURST Refresh
        MOVEA.L #COL,A3
        MOVEA.L #B_READ,A4
        MOVE.W #ROWMAX,D2
        MOVE.W #$10,(A2)

FRESH2: MOVE.B #0,(A2)         ;set RAS low
        MOVE.B #0,(A3)
        MOVE.B (A4),(A5)       ;refresh entire array once
        MOVE.B #1,(A2)        ;set RAS back to high
        MOVE.B #1,(A3)
        ADDA.L #2,A2           ;increment row address
        ADDA.L #2,A3
        ADDA.L #2,A4
        SUBQ.W #1,D2           ;decrease counter by 1
        BNE FRESH2
        MOVE.B #10,D3         ;reload 10 row counter

SKIP:   SUBQ.B #1,D0           ;decrease ROW counter by 1
        BNE READ

        MOVE.L D7,D2           ;Send count of 1's versus 0's
        LSR.L #8,D2
SBY3:   BTST.B #06,R0R
        BEQ SBY3
        MOVE.B D2,R7W
SBY4:   BTST.B #06,R0R
        BEQ SBY4
        MOVE.B D7,R7W
        RTS

```

```

;-----

```

```

;MAIN: -input      : ---
MAIN
;      -output     : ---
;      -subroutines :HPIBINIT, GETCHR, VERIFCHR, ROMRAM, PITINIT

```

```

;      -reg. affected : ---
;      -description   : this is where the program really starts
MAIN:   BSR ROMRAM      ;copy the program from the EPROM into the
RAM
        JMP RAMBASE+JUMPER ;transfer program control to RAM
JUMPER: BSR HPBINIT     ;initialize HPB bus
        BSR PITINIT     ;initialize PIT port, output values
undefined
        CLR.B PWR       ;indicate power up status = not intialized
START:  BSR GETCHR      ;get a command character from the HPB
        BSR VERIFCHR    ;verifies the char sent, and execute
command
        BRA START       ;loop back to start
        END
;----- (C)1989 MIG
STUDIOS --

```

Appendix D - Sample plots
 Appendix D1 - Plot of ALU circuit diagram

Appendix D2 - Plot of ALUS simulation

END

'21-07-94

FIN