1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

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

Produit en collaboration avec

la Bibliothèque nationale

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

Design and Implementation of a Multiprocessor for DSP and Communications Systems Algorithm Evaluation

George Nicholas Yaremchuk, B.Eng.

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Engineering.

Department of Electronics
Faculty of Engineering
Carleton University
Ottawa, Canada

May 1993

© Copyright George N. Yaremchuk 1993

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

Author's - Auteur's address

Author's - Auteur's address

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 0-315-84128-1

Canada

Name George D. Yurumuk

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Electronics and Electrical

SUBJECT TERM

0344

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0222
Art History 0277
Cinema 0200
Dance 0278
Fine Arts 0257
Information Science 0223
Journalism 0292
Library Science 0299
Mass Communications 0208
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0227
Early Childhood 0518
Elementary 0524
Finance 0227
Guidance and Counseling 0519
Health 0680
Higher 0245
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0298
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0214
Secondary 0533
Social Sciences 0534
Sociology of 0240
Special 0529
Teacher Training 0530
Technology 0210
Tests and Measurements 0288
Vocational 0247

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern 0291
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology 0323

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Marketing 0385
Canadian Studies 0501
Economics 0503
General 0505
Agricultural 0508
Commerce Business 0509
Finance 0510
History 0511
Labor 0358
Theory 0366
Folklore 0351
Geography 0578
Gerontology 0578
History 0578
General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General 0616
International Law and Relations 0617
Public Administration 0814
Recreation 0452
Social Work 0626
Sociology 0627
General 0938
Criminology and Penology 0631
Demography 0628
Ethnic and Racial Studies 0629
Individual and Family Studies 0630
Industrial and Labor Relations 0700
Public and Social Welfare 0344
Social Structure and Development 0709
Theory and Methods 0999
Transportation 0453
Urban and Regional Planning 0453
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy 0475
Animal Culture and Nutrition 0475
Animal Pathology 0475
Food Science and Technology 0475
Forestry and Wildlife 0475
Plant Culture 0475
Plant Pathology 0475
Plant Physiology 0475
Range Management 0475
Wood Technology 0475
Biology 0405
General 0285
Anatomy 0308
Biostatistics 0309
Botany 0310
Cell 0311
Ecology 0312
Entomology 0313
Genetics 0314
Immunology 0315
Microbiology 0316
Molecular 0317
Neuroscience 0318
Oncology 0319
Physiology 0320
Radiation 0321
Veterinary Science 0322
Zoology 0323
Biophysics 0324
General 0325
Medical 0326

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0295

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetrics and Gynecology 0354
Occupational Health and Therapy 0381
Ophthalmology 0371
Pathology 0419
Pharmacology 0572
Pharmacy 0382
Physical Therapy 0573
Public Health 0574
Radiology 0575
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0749
General 0486
Agricultural 0487
Analytical 0488
Biochemistry 0738
Inorganic 0490
Nuclear 0491
Organic 0494
Pharmaceutical 0495
Physical 0754
Polymer 0405
Radiation 0605
Mathematics 0986
Physics 0606
General 0608
Acoustics 0748
Astronomy and Astrophysics 0607
Atmospheric Science 0798
Atomic 0759
Electronics and Electricity 0609
Elementary Particles and High Energy 0610
Fluid and Plasma 0752
Molecular 0756
Nuclear 0611
Optics 0463
Radiation 0346
Solid State 0984
Statistics 0984

Applied Sciences

Applied Mechanics 0346
Computer Science 0984

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 0551
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal 0790
System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

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

Design and Implementation of a Multiprocessor for DSP and Communications Systems Algorithm Evaluation

George Nicholas Yaremchuk, B.Eng.

A thesis submitted in partial fulfilment of the requirements for the degree
of Master of Engineering.

Thesis Supervisor

Chair, Department of Electronics

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

May 1993

Abstract

It is proposed to create a test bed to ease algorithm and hardware development tasks. The advances in VLSI have created a situation in which complex signal processing algorithms are considered even for high speed, high computational complexity applications (HDSL, speech processing, and mobile communications can serve as examples). For these applications for which only an incomplete analytical performance evaluation can be made, the problem of signal processing algorithm verification becomes a bottleneck in both research and hardware development stages. A programmable hardware algorithm evaluation system capable of operating at or near real-time for the data rates of interest is required. A multiprocessor system based on a slotted ring bus communication scheme has been adopted as a compromise between interconnection bus complexity and bus transfer efficiency. Review of selected DSP system algorithms has been used to verify the design choices. Then the hardware was implemented based on the TMS320C30, tested, and benchmark results given. The test bed discussed here can serve as a test bed for complex (digital) signal processing algorithms. The speed up in evaluation time creates, for the user, nearly "a final product" testing situation providing results at or near real time.

Acknowledgments

There are many people that deserve my gratitude in making this thesis possible. First, many thanks to my wife Gail for her faith and persistence in "getting the job done", to my parents for their support and patience, and (greatly understated) thanks to professor Tad Kwasniewski for his advice, patience, and perseverance throughout this ordeal. Appreciated is the advice and assistance rendered by professors David Falconer, Rafik Goubran, Miles Copeland, and kudos to Barb, Denis, and Guy for their endless help and humouring of my requests. In addition, I would like to acknowledge the excellent advice of professor John Knight and the graduate students with which I have had the pleasure of making acquaintance.

As to those that had complete and utter faith in my "documentation skills" (or lack thereof), I have but to say...

"Nah, Nah, Nah, Nah, Naaaaah!" (Yes, that does mean you Paul and Larisa.).

Finally, I would like to dedicate this thesis to those that were instrumental in my formative years: my parents Elias and Irene Yaremchuk, and my god-parents Peter Michi-alyshin and Sonya Klymenko.

Table of Contents

Abstract	iii
Acknowledgments	iv
List of Figures	ix
List of Tables	xi
Nomenclature	xii
 Chapter 1 Introduction	
1.1 Statement of the Problem	1
1.2 Motivation for the Algorithm Test Bed Development	2
1.3 Areas of Application	2
1.4 The Importance of the Multiprocessor Intercommunication Architecture	3
1.5 Thesis Approach	3
 Chapter 2 Literature Overview	
2.1 Introduction	5
2.2 DSP Algorithm Data Flow	5
2.2.1 Applicable DSP Algorithms	6
2.2.2 Algorithm Subblock Intercommunication Considerations	6
2.2.2.1 Synchronous Data Flow (SDF)	7
2.2.2.2 Data Flow Concurrency	8
2.2.2.3 Data Flow Characteristics	8
2.2.2.4 Computation/Communication (R/C) Ratio	11
2.2.2.5 Data Flow Synchronization	12
2.3 Multiprocessor Architectures	13
2.3.1 Interconnection Schemes	13

Table of Contents

2.3.1.1	Global Bus Topologies	13
2.3.1.2	Local Bus Topologies	15
2.3.1.3	Multilevel Switched-Network Topologies	16
2.3.1.4	Crossbar Interconnection Topologies	18
2.4	DSP Oriented Multiprocessor Hardware Implementations	18
2.4.1	Commercially Available Systems: The ASPI "Banshee" Multiprocessor	19
2.4.2	Research Systems	20
2.4.2.1	The "HPTS"	21
2.4.2.2	The "PT8"	21
2.4.2.3	The Texas Instruments "Odyssey"	22
2.4.2.4	The "MUPSI" System	23
2.4.2.5	The DSP.* Multiprocessor	24
2.4.2.6	The SMART Architecture	25
2.4.2.7	The RAP Multiprocessor for ANN Simulation	28
2.4.2.8	The Berkeley/Dolby System	29
2.5	Conclusions	30

Chapter 3 Candidate Algorithms

3.1	Introduction	32
3.2	DSP Application Algorithm Examples	32
3.2.1	FIR Intra-algorithm Data Flow	33
3.2.2	The HDSL Transmitter/Channel/DFE Receiver	36
3.2.3	MEL-Cepstrum Speech Pre-processing	39
3.3	Conclusions	40

Chapter 4 Proposed Interconnection Architecture

4.1	Introduction	42
4.2	Bus Architecture Comparison	42
4.2.1	Global Bus Topologies	43
4.2.2	Ring Bus Topologies	43
4.2.3	Other Bus Topologies	43
4.2.4	The Selected Bus Architecture	45
4.3	Slotted Ring Bus Description	46
4.3.1	Barrier Synchronization	48
4.3.2	Comparison to Existing Systems	49
4.3.3	Additional Hardware Requirements	50
4.4	Conclusion	51

Table of Contents

Chapter 5 The Hardware Design

5.1	Introduction	52
5.2	DSP Processor Comparison	52
5.3	Processor Choice	56
5.3.1	Why Choose the C30	56
5.3.2	C30 Detailed Description	57
5.4	Slotted Ring Bus Features and Protocol	59
5.5	Slotted Ring Bus Hardware Design	60
5.5.1	Bus Data Transfer Timing	61
5.5.2	IOP Bus Interface Controller	62
5.5.3	Multiprocessor System Backplane and Controller Module	63
5.5.4	Inter-processor Synchronization Hardware	64
5.5.4.1	Bus Interrupts for Barrier Synchronization	65
5.5.4.2	External Flag Barrier Synchronization	66
5.5.4.3	Dual Port RAM (DPRAM) Signalling Synchronization	67
5.5.4.4	The Signalling Bus	68
5.5.5	Bus Performance	69
5.6	Processing Element (PE) Features	72
5.6.1	Central Processing Unit (CPU)	73
5.6.2	Local Memory	73
5.6.3	Trace Buffer	73
5.6.4	Interrupt Control	74
5.6.5	Parallel Input/Output Control Register	75
5.6.6	Parallel C30 Host Interface	75
5.6.7	Interprocessor Communication Bus	75
5.7	Conclusion	76

Chapter 6 Hardware Verification and Performance Evaluation

6.1	Introduction	78
6.1.1	Functional Verification	78
6.1.1.1	PE Testing	79
6.1.1.2	IOP Bus Interface Testing	79
6.1.1.3	Signalling Bus	80
6.1.2	Hardware Performance Verification	80
6.2	Algorithm Performance Evaluation	80
6.2.1	Program Synchronization	81

Table of Contents

6.2.2	MEL-Cepstrum Speech Processing Front End	82
6.2.3	HDSL Transmitter-Channel-DFE Receiver Simulations	83
6.2.4	Other Execution Examples	86
6.3	Conclusion	87

Chapter 7 Conclusions

7.1	Conclusions	88
7.2	Future Work	89

References	90
-------------------	-----------

List of Figures

Figure 2.1	Data Flow Concurrency	9
Figure 2.2	Local Interconnect Architectures	16
Figure 2.3	Three Level Shuffle Exchange Multilevel Interconnect	17
Figure 2.4	Four Processor, Three Memory Crossbar Structure	19
Figure 2.5	The Banshee MPI Block Diagram	20
Figure 2.6	HPTS Partial System Block Diagram	21
Figure 2.7	The MUPSI Data Exchange Unit Block Diagram	23
Figure 2.8	The SMART Multiprocessor Block Diagram	25
Figure 2.9	Bus Interface Structure of a RAP Processing Node	29
Figure 3.1	FIR Filter Data Flow Diagram	33
Figure 3.2	Pipelined Partitioning of the FIR Filter	34
Figure 3.3	Parallel Partitioning of the FIR Filter	34
Figure 3.4	Retimed Pipeline Partitioned FIR Filter	35
Figure 3.5	FIR Activity Chart and Communication Volume	36
Figure 3.6	HDSL Transmitter, Channel Simulator, and DFE Receiver	37
Figure 3.7	Pipelined HDSL Data Transmitter and Channel Simulator	38
Figure 3.8	Mel-Cepstrum Algorithm Data Flow Diagram	39
Figure 4.1	Single and Chordal Five PE Ring Buses	45
Figure 4.2	Slotted Ring Structure for Three Processing Elements	46
Figure 4.3	Bus Input/Output Processor (IOP) Block Diagram	48
Figure 4.4	Address Tag Comparison Block Diagram	48
Figure 4.5	Barrier Synchronization Timing Example	49
Figure 5.1	C30 Multiprocessor Processing Element (PE) Block Diagram	58
Figure 5.2	Three PE Ring, with IOP Structure Shown	60
Figure 5.3	Bus Controller Components	62
Figure 5.4	MPU System Control Module Block Diagram	64
Figure 5.5	Bus Interrupt Block Diagram	65
Figure 5.6	External Flag Synchronization Bus	66
Figure 5.7	Signalling Synchronization Bus Structure	68
Figure 5.8	Detailed Signalling Bus Interface Block Diagram	69
Figure 5.9	Processing Element Block Diagram	73

List of Figures

Figure 5.10	Parallel C30 Emulator Interface Bus	76
Figure 6.1	Software Development Cycle	81
Figure 6.2	Program Synchronization of Data Flow	82
Figure 6.3	MEL Cepstrum Block Diagram Partitioned for Two PEs	82
Figure 6.4	HDSL Transmitter-DFE Receiver Partitioned Over Two PEs	84
Figure 6.5	Fully adaptive HDSL Transmitter-DFE Receiver Partitioning (4 PEs)	85

List of Tables

Table 2.1: Vestigial Sideband Example Execution Results	24
Table 2.2: "SMART" System DSP Application Simulations	26
Table 2.3: "SMART" System Actual Results	28
Table 4.1: Intercommunication Time Comparison for Single and Chordal Ring Buses	45
Table 5.1: DSP Microcomputer Comparison	53
Table 5.2: Latency for Furthest Intercommunication Path in a N PE System	70
Table 5.3: Avg. Intercommunication Latency for Varying Number of PEs in a System	71
Table 5.4: Peak Data Transfer rate for Various Number of PEs in a System	72
Table 5.5: Adjacent PE Data Transfer Latency	72
Table 5.6: C30 Interrupt Processing Latency	75
Table 6.1: Execution Time Comparison for the Mel-Cepstrum Algorithm	83
Table 6.2: Execution Time Comparison for the Simple HDSL Algorithm (2 PEs)	84
Table 6.3: Execution Time Comparison for the Full HDSL Algorithm (4 PEs)	86

Nomenclature

AFIR	Adaptive FIR
ARMA	Auto Regressive Moving Average
BER	Bit Error Rate
C	refers to the C programming language
C30	Texas Instruments TMS320C30 DSP CPU IC
C40	Texas Instruments TMS320C40 DSP CPU IC
CPU	Central Processing Unit
DAFIR	Delayed Adaptive FIR
DFE	Decision Feedback Equalizer
DFT	Discrete Fourier Transform
DLMS	Delayed Least Mean Squares
DSP	Digital Signal Processing
DTW	Dynamic Time Warping
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
HDSL	High-rate Data Subscriber Loop
IC	Integrated Circuit
IIR	Infinite Impulse Response
IOP	Input/Output Processor
ISDN	Integrated Services Data Network
LMS	Least Mean Squares
PE	Processing Element
TI	Texas Instruments Ltd.
VLSI	Very Large Scale Integration
VQ	Vector Quantization

Chapter 1 Introduction

1.1 Statement of the Problem

The development of new digital communication systems relies heavily on the progressive development of *Digital Signal Processing* (DSP) algorithms and their interaction in various combinations as DSP systems. The communication systems are being used at increasingly higher bit/ baud rates and/or are of higher computational complexity to meet ever increasing needs. The typical system operates on an essentially 'infinite' length input data vector(s), upon which a series of operations (algorithms) is applied, either on a item by item or block basis. To ensure that the proposed system will meet the specified performance criterion, a performance evaluation must be conducted.

A communication system performance evaluation often relies on estimates of statistical properties, which are increasingly obtained by simulation of the system. An example of a statistical performance parameter often used in data communication systems is the *Bit Error Rate* (BER: stated as the probable number of bit errors that will occur per symbol transmitted e.g. 11 errors/1e10 symbols transmitted = 1.1×10^{-9} BER). The time required to simulate a system for a given BER performance probability is an exponentially proportional function of that BER probability. This is because to obtain the next point on the BER curve, normal practice is to utilize a multiple of the number of bits transmitted used to obtain the last point, e.g. 2 or 10 times as many points. When this is combined with the increasing complexity of algorithms that can be implemented (due to density advances in VLSI that will be used in the final application), extremely long input vectors are required for valid test results at low BERs.

The algorithm development and evaluation can be performed by software techniques using either a system level simulator or dedicated software code, or by a hardware prototype implementation (either in custom silicon and/or using off-the-shelf components). Due to the increased system complexity, software simulation is increasingly becoming prohibitively slow, and precludes exhaustive testing to the levels required by

low BERs common to digital communication systems. The trial of new what-if scenarios is limited due to the large amount of time required to obtain new results. Simplifications in the simulation to decrease the simulation time lead to decreased confidence in the final result. The dedicated hardware approach can achieve real time or near real time performance, but at a relatively high cost in time and effort. The hardware approach is prone to having multiple prototypes constructed as the design changes, and again the number of what-if scenarios that can be tried is limited by the hardware change time and cost. In current practice it is usual to combine the two techniques; software to do a limited initial analysis of the algorithms and then hardware to prototype the system for extensive testing at real time or near real time rates. It is felt that this method is rapidly becoming inadequate as communication system complexity increases. These inadequacies cover such items as over specification of the system to ensure the performance criterion are met (due to low confidence in the performance evaluation results), final systems not meeting performance objectives, and the associated costs with the above.

The topic of this thesis is to develop a programmable hardware tool that is suited to modelling DSP communication systems that are composed of less complex DSP sub-blocks, i.e. an algorithm test bed. Furthermore, the speed of the algorithm simulation must be at or near real-time for baud rates of current interest to allow full evaluation of the communication hardware model, to increase the speed and accuracy with which the final system (e.g. VLSI) is implemented.

1.2 Motivation for the Algorithm Test Bed Development

This tool could be of interest to both the academic and industrial communities for similar but yet varying reasons. In academic research, it allows a more thorough and/or broader investigation of new algorithms and combinations of these within a given research interval. Industry requires a similar capability in research and development, as well as the capability to fine tune algorithms for better field performance within a smaller time frame without resorting to multiple prototypes. This fine tuning of the algorithm is usually associated with the reduction in final product complexity and thus cost, while still meeting performance criterion.

1.3 Areas of Application

A proposed area of research is in the field of Integrated Services Digital Networks

(ISDN) subscriber loops at increasingly higher bit rates. For algorithms used in 800 Kbit/sec subscriber loop (HDSL) applications the processing rate would be 400 Kbaud, resulting in a baud period of 2.5 microSeconds. In this application, the actual channel is typically simulated because the amount of storage required to pre-calculate the required number of symbols for low BER applications is impractical. The algorithm test bed has also been proposed for use in the testing of complex speech processing application such as compression and recognition.

This baud period in the former application (2.5 μ S) is not lengthy in terms of the number of instructions that can be executed by the fastest of the current generation of DSP oriented processors. For example, the Texas Instruments TMS320C30 can execute approximately 42 instructions (at 60 nS per instruction), the Texas Instruments TMS320C40 can execute approximately 63 instructions (at 40 nS per instruction), and the Motorola DSP56000 can execute approximately 26 instructions (at 97.5 nS per instruction). This implies that algorithms of even moderate complexity cannot be implemented at or near real-time on a single processor, therefore the use of multiple processors is required.

1.4 The Importance of the Multiprocessor Intercommunication Architecture

The driving force in the design of a multiprocessor system is the processor intercommunication structure. For the target of real-time or near real-time operation, a fixed communication overhead consumes a proportionately larger percentage of the total computational capability of the system as the data (baud) period decreases. Intuitively, a necessary requirement for a multiprocessor is that the intercommunication architecture and implementation should operate efficiently for the range of applications of interest. Restated, it is necessary to reduce the processor intervention required for intercommunication to a bare minimum (either in direct action or passive waiting), so that it can spend most of its time computing the problem, not moving data about.

Other criterion for the selection of the intercommunication structure are of practical nature. The practical scope of the design and construction of such a system must be within the resources of an individual master's thesis.

1.5 Thesis Approach

Chapter 2 is a literature review that encompasses three areas. First, a study of the

data flow of *digital signal processing* (DSP) algorithms was undertaken to define the algorithm subblock intercommunication characteristics. The general bus architectures commonly used in multiprocessor systems were then reviewed. Then an overview is presented of the bus architectures used in commercial and research multiprocessor systems that are targeted towards DSP algorithm execution.

In chapter 3, DSP communication algorithms of interest are introduced, and their data flow characteristics briefly reviewed and compared to the characteristics stated in chapter 2.

The next chapter, chapter 4, the suitability of various previously reviewed multiprocessor intercommunication architectures to efficiently implement the stated data flow characteristics is considered. A suitable bus architecture that improves upon the deficiencies found is then proposed and described.

Chapter 5 provides a detailed account of the multiprocessor hardware design. Various DSP processors ICs currently available are considered, and one chosen for use in this design. The design and performance parameters of the bus architecture are given. Additional design details of the *processing element* (PE) are given.

The actual hardware was then designed, built and tested. A set of low level software tools was created to aid in testing and programming of the system. Then two of the previously presented example system level algorithms were coded and benchmarked to evaluate the test-bed's suitability in meeting its original objectives.

Chapter 6 describes the testing of the hardware, and the benchmarking of these examples on the algorithm test bed. The benchmark results are presented.

The conclusions of the thesis and suggestions for future work are presented in chapter 7.

Chapter 2 Literature Overview

2.1 Introduction

To implement a successful multiprocessor system design, there are a number of areas that must be studied. First the ramifications of the typical algorithms that the system is expected to execute must be determined. In this case the algorithms will be DSP applications that utilize basic DSP subblocks such as FIR filters, FFTs, etc. The interconnection architectures that are available to allow data transfer between processing elements should be considered in light of their advantages and disadvantages. The general categories that will be considered are global bus, local interconnect (ring type), and multilevel interconnects. Finally, the review of some existing systems constructed to date for similar applications should provide some aid in evaluation of various approaches.

2.2 DSP Algorithm Data Flow

The DSP algorithms can be subdivided into two classes: those that are elemental building blocks that are usually considered as indivisible entities (atomic blocks [Lee87c] or *Fine Grain Data Flow* – FGDF [Ston87a]), and subsystem level blocks that are a combination of the atomic blocks. The subsystem blocks are then considered as elemental building blocks that are interconnected to solve system level problems. This level of abstraction is referred to as *Large Grain Data Flow* (LGDF) [Lee87c] or *coarse Grain Data Flow* (CGDF) [Ston87a]. Our research interests lie with the CGDF level of abstraction, thus the resulting multiprocessor must execute this level of block interconnection efficiently. The subsystem blocks will usually be assigned to run in their entirety on one of the processors in the system (i.e. no intra-block partitioning is used). The inter-block data flow in system algorithms will be examined to determine the constraints imposed on the hardware solution.

2.2.1 Applicable DSP Algorithms

There are a wide range of DSP algorithms that are suited for a multitude of tasks i.e. the same problem can be solved using many different techniques. The result is a very large number of algorithms that are variations of more basic or mainstream algorithms. When the multiprocessor system is used as a research tool, new algorithms and variations of previous ones would be developed, making it difficult to predict how the algorithm characteristics would affect the multiprocessor design. However, the basic algorithms can be examined as to their data flow requirements, and the resulting constraints applied to the multiprocessor hardware design. Some of the LGDF subblocks encountered [Fal-c82a][Lin90a][Lin90b][Moha90a][Muel79a] are listed. These are:

- . Finite Impulse Response (FIR) Filters
- . Adaptive FIR Filters (including echo cancellers, decision feedback equalizers, etc)
- . Auto Regressive Moving Average (ARMA) Filters
- . Infinite Impulse Response (IIR) Filters
- . Discrete/Fast Fourier Transform (DFT/FFT)
- . Convolutional Codes
- . Mel Cepstrum speech analysis front ends
- . Dynamic Time Warping (DTW)
- . Vector Quantization (VQ)
- . Viterbi Decoder

Those deemed more important (most widely used) will be discussed in the context of effect on multiprocessor design. Of the above, the FIR, AFIR, and FFT have long been the workhorses of DSP systems.

2.2.2 Algorithm Subblock Intercommunication Considerations

There must be some criterion for evaluating how algorithm would affect the choice of a particular multiprocessor architecture. Since we are considering a MIMD type of multiprocessor (Multiple Instruction stream-Multiple Data stream; see section 2.3 *Multiprocessor Architectures*. for further explanation), the main concern is to evaluate the effect that interprocessor communication will have on the selection of the multiprocessor interconnection architecture. The algorithm inter-block data flow characteristics of the DSP systems considered here are:

- . restrictions on the data flow diagram for typical DSP flow graphs
 - . synchronous data flow
 - . determinism of communications
- . data flow concurrency
- . data flow characteristics
 - . direction
 - . data flow latency
 - . locality of communication between PEs
 - . the effect of non-local feedback loops
 - . data processing method (sample or block basis)
 - . the temporal distribution of the intercommunication (data traffic congestion)
- . algorithm computational complexity verses communication requirements (R/C ratio)
- . data flow synchronization

Many of these same considerations are also applicable to the implementation of DSP algorithms in VLSI integrated circuits [Deca89a]. This is due to the similarity of algorithm subblock intercommunication restrictions in a VLSI physical layout design, and in interconnecting multiprocessors on or between printed circuit boards. For this reason, literature that studies the partitioning and rearranging of DSP algorithms for implementation in VLSI can also be applicable to implementing them on multiprocessors, especially those architectures that are locally connected.

2.2.2.1 Synchronous Data Flow (SDF)

Data flow diagrams are a natural method of describing DSP systems, and have been used by many block level graphical DSP design environments [Lee87b] to help design and simulate DSP algorithms. In a data flow graph, the algorithm is described as a directed graph (see *Figure 2.1*) where the node represent computational functions and the connecting arcs are paths that sequences of data (alternatively called packets, tokens, or samples) flow upon.

Synchronous data flow, as described by Lee *et al* [Lee87a][Lee87b][Lee87c] is a special case of data flow. The major assumption to restrict the scope of the general data flow concept is that the amount of data flowing along any arc is fixed a priori to run-time i.e. the number of tokens consumed by each node input, and the number of tokens produced by each node output are constant for each invocation of the node. Multiple sample

rates are permitted, however all sample rates must be rational multiples of each other to preserve the synchronous nature of the SDF. For DSP systems the data input and output are considered to be semi-infinite sequences of data. The computation performed on these data input sequences is considered to be periodic (called a *periodic schedule*, in [Lee87c]), i.e. each sample or block of samples has the same computation performed on it. Further more, it is stated that run-time conditional control flow within a block is rare in signal processing [Lee87a].

One of the main advantages of SDF systems is that because the data flow quantity is static, the partitioning of the system onto a multiprocessor can be performed statically (prior to run time), simplifying the implementation and the run-time overhead of dynamic scheduling.

2.2.2.2 Data Flow Concurrency

In general, the concurrency in the algorithm can be exploited by partitioning in either a pipelined or a parallel sense [Ston87a]. A pipelined organization is illustrated in Figure 2.1 b). Each PE processes the data in whole and then passes it on to the next processing block. A parallel partitioning approach is illustrated in Figure 2.1 a). Each processing block operates on the same incoming data or a portion of it in parallel with all of the other blocks. As well, these two methods can be intermixed in one data flow partitioning diagram to suit the flow of the algorithm data.

The algorithms can be arranged and re-timed in different ways [Kung82a] [Lawr83a] [Meye88a] [Potk90a] so as to allow efficient mapping onto hardware that has been implemented to support one of the above organizations. However, many DSP systems tend to be modelled by the pipelined concurrency paradigm [Bolc87a][Deca89a], i.e. they adhere to linear synchronous data flow. An intercommunication architecture that models these characteristics may be the most suitable for this application.

2.2.2.3 Data Flow Characteristics

Another characteristic often observed in DSP systems is unidirectional data flow [Bolc87a][Deca89a]. The DSP algorithms can often be rearranged to suit a linear pipelined type of data flow by changing the calculations performed (but kept functionally equivalent) [Theo90a] or retiming the algorithm, similar to techniques used for systolic arrays [Kung82a]. Bidirectional data flow can be represented as two unidirectional data paths.

Figure 2.1 Data Flow Concurrency

Linear data flow graphs imply that data communications most often occurs between adjacent subblocks, and less often between non-adjacent processors [Bolc87a]. This locality of intercommunications indicates that for efficient performance of the system, it is more important to have fast communication between adjacent processors, and less so the further apart the processors are in the interconnection architecture.

A side-effect of unidirectional linear data flow is that latency in data intercommunications can be tolerated to a large degree[Bolc87a]. In addition, latency effects can be ignored due to the end use of the multiprocessor system as a DSP algorithm test bed. This use as a modelling vehicle removes any real-time latency restrictions. The initial output is delayed by the latency amount of processing steps, but then output samples are available on subsequent processing steps. The retiming of some algorithms to better suit the linear pipelined data flow [Kung82a] usually introduces additional latency, thus this tolerance is important.

Large feedback loops in the application algorithm violate the above assumptions. The data flow is non-linear, in that the results from a subblock calculation are fed back to previous stages. If the subblocks are on physically different PEs, which results in communication latency, then data throughput can be affected. This is because the data must be calculated by the source of the feedback data, sent to and received at the input of the block receiving the feedback, and used to complete its calculations. All of these calculations must occur within a single data processing interval, and thus the intercommunication latency must be added to the computational period. For unavoidable feedback loops, the inter-processor communication time must be kept as short as possible. However there are

important cases in which this feedback loop constraint can be minimized. In the case of the adaptive FIR filter, the above is true in the strict sense for the calculation of the error signal and its use in updating the tap coefficients. In practice it has been found that updating the tap coefficients one or even a few baud intervals later has no appreciable effect on filter performance [Dar188a] [Haim90a] [Kaba83a] [Long87a]. This suggests a limit on the maximum intercommunication latency of one baud interval is tolerable, but this must be reviewed for each algorithm. The previously mentioned retiming of the algorithm can also be used to help reduce the effects of overall feedback loops. Algorithms can also be rearranged in form but not function to alleviate the effect of feedback loops.

The SDF model indicates that the quantity of data flow between blocks is known a priori to run time. Further, many common signal processing algorithms are run-time invariant [Lee87a], i.e. there are no conditional control flow decisions at run time. Together these imply that the intercommunication between blocks is generally fully deterministic in the quantity and time of occurrence. Thus the temporal distribution can be surmised prior to run time, and the peak data traffic determined.

It has been previously established that many DSP applications operate in a periodic manner on the input data, and the temporal distribution of intercommunication is relatively fixed. These independent processor computations are often synchronized to begin computation at about the same time. In partitioning of an algorithm, the computations can be similar or the same on different processors, and thus access the external resources at approximately the same time [Ston87a]. Intuitively, one expects a computation to consume input data near the beginning of the computational cycle, and produce output data near the end of the computational cycle (i.e. data in at the start of a processing cycle, data output near the end). In this sense, two different subblocks running on two separate processors can appear similar in their temporal intercommunication patterns. This implies that the intercommunication traffic will occur in periodic bursts with relatively little traffic in intervening periods. In the results for DSP algorithms run on the SMART system [Koh88a] when configured for fully shared buses, the bus was clearly saturated but the percentage of bus use was well below 100% (see Table 2.2, *SMART: System DSP Application Simulations*, on page 26). This implies that bus accesses occurred in bursts during which the bus saturated, intermixed with intervals of bus inactivity, which when combined result in a less than full percentage of bus use.

The data processing granularity of DSP algorithms occur in two basic forms. Data is often processed either on a sample by sample basis (e.g. FIR filters) or on a block by block basis (e.g. FFTs). This can effect the severity of bus contention when blocks of data are utilized, because most often it is intercommunicated in whole blocks. This in turn can

magnify the previously mentioned burst like intercommunication nature, causing greater peak contention for intercommunication bandwidth.

2.2.2.4 Computation/Communication (R/C) Ratio

A basic performance measure of multiprocessor systems is the R/C ratio [Ston87a]. The **R** is the amount of run-time required i.e. the computational period, and **C** is the time required to complete the inter-communications resulting from the computation. When dividing a task onto a multiprocessor, it is generally recognized that the intercommunication overhead is the limiting factor in the expected performance increase. In terms of the R/C ratio, if the ratio is small (a large percentage of time is spent communicating) then the gains of exploiting concurrency are not high. Conversely if R/C is large then the relative percentage of communication time is small and a larger expected gain can be expected. The R/C ratio is also a measure of the task granularity. For coarse-grain concurrency (CGDF, see section 2.2.2.2 *Data Flow Concurrency*.) the R/C ratio is high, and for fine-grain concurrency (FGDF) it is low. Thus the gain of exploiting concurrency is expected to be higher for the CGDF case. Previously we stated that the purpose of this system is to simulate coarse-grain problems, and as such the R/C ratios are generally higher; consequently a relatively large incremental increase in performance is expected from exploiting concurrency.

A potential problem of CGDF partitioning is having too large a grain when partitioning an algorithm. If no intra-block partitioning is utilized, then it may be difficult divide the blocks evenly between processing elements so that there is a low amount of computational idle time in the system. This can be potentially be a problem in typical systems of interest to us. For example, when simulating telephone line channels and then removing the resulting echo in ISDN HDSL simulations [Moha90a], the length of the adaptive FIR filters can be 300 to 400 taps (i.e. the pulse response of the effects of the line can be measured for up to 600 or more band periods, and are considered significant to 300-400 baud periods). The processing time of these can be much longer than of other system sub-blocks, which results in unbalanced computational loading of the processor elements if no intra-block partitioning is utilized. This was a major factor in the MUPSI system not achieving the maximum theoretical speedup [Bolc87a] (see The MUPSI System, on page 23).

A method that is widely used to effectively increase the R/C ratio is to reduce the **C** value by utilizing overlapping communication with computation. This can be done by introducing hardware to concurrently process intercommunications. Common examples

are Direct Memory Access (DMA) units, I/O processors such as the Intel 8089 [Haye88a], and intelligent peripherals now commonly found in generic IBM-compatible PCs. Additional hardware techniques such as dual buffered and/or ported memories, and FIFOs allow more processing/communication concurrency by decoupling the interaction between these two tasks. However, the intercommunication path architecture chosen for the multiprocessor can affect the amount of overlap possible [Ston87a]. A shared bus forces communication idle time during contention, and processor synchronization schemes usually impose processor idle time. The use of some type of independent I/O processing hardware was used in some implemented local interconnect type multiprocessor systems [Bolc87a][Kahr88a][Koh88a].

2.2.2.5 Data Flow Synchronization

The synchronization of multiple concurrently executing programs, each on separate processors, is vital to the successful utilization of a multiprocessor system. Stone [Ston87a] states that synchronization programming is *probably the most difficult and error-prone type of programing that exists..* This is due to the difficulty in trying to anticipate and understand the many possible simultaneous actions of the multiple processors. The synchronization programming process is further complicated by the dependency upon hardware provisions available to support it. Common examples of such hardware are the indivisible read-modify-write instructions (i.e. bus locking) available on processors and many global buses, and hardware interrupt generating semaphore memory locations in dual port RAM ICs. The level of complexity of the synchronization is also a function of the interconnection architecture chosen for the multiprocessor. It is intuitively simpler to synchronize data flow between two locally connected processors, than for ten processors accessing a global memory on a shared bus.

Koh *et al* [Koh88a] state that of the two most common synchronization types: exclusive access and barrier, that barrier synchronization is adequate for most cases found in DSP systems. Barrier synchronization places an execution barrier in each processor's program that cannot be passed until all programs have reached the barrier. Processors whose programs have reached the barrier idle until the last (slowest) processor program finally reaches the barrier, whereupon all processors then proceed with the next phase in computation. This model fits the periodic nature of many DSP algorithms previously mentioned in Section 2.2.2.1.

2.3 Multiprocessor Architectures

A basic classification of multiprocessor architectures is between single-instruction stream, multiple-data stream (SIMD) and multiple-instruction stream, multiple-data stream (MIMD) [Ston87a]. The SIMD processing elements operate in lock-step using the same instruction broadcast to all PEs, but on different data input streams. In other word all data streams are processed in parallel in exactly the same manner. The class of problems that fit this type of operation is limited, and in general DSP algorithms do not fall within its scope. A MIMD architecture is characterized by several independent processors that are interconnected in some manner. Each processor is capable of executing an independent (from the other processors) program. The execution of programs on individual processors is synchronized in some manner so that data can be passed between them and computed on. This paradigm is applicable to a wider range of problems, and is the one most often pursued when designing a multiprocessor system.

2.3.1 Interconnection Schemes

There are a wide variety of multiprocessor interconnection schemes, but many are variations of four basic topologies [Haye88a][Ston87a]:

- . global (shared) bus
- . local bus
- . multilevel switched-network
- . crossbar interconnection

The crossbar can be considered an extension of the global bus topology, but is usually treated a a different scheme in the literature. The attributes, advantages and disadvantages of each bus will be discussed in turn.

2.3.1.1 Global Bus Topologies

The global bus is perhaps the conceptually simplest of all multiprocessor interconnection topologies. Generally each processor has local resources (memory, I/O, etc.) but intercommunicates with other processors in the system via a single shared bus. The intercommunication may take place by a shared global resource such as a global memory (shared-memory system [Haye88a]), or by access to another processors local resource such as dual-ported memory or a FIFO (distributed-memory system). The main point is that only one processor can have access to the shared bus at any one given time. To gain

access to the bus, some type of bus request – bus grant arbitration scheme is utilized. In shared-memory systems there is typically some type of bus lock mechanism to allow a processor to have guaranteed access to the bus over several consecutive bus transactions. This is required to implement reliable synchronization of the use of global bus resources (e.g. test-and-set access to a global memory location to implement semaphore type inter-processor synchronization). In general distributed-memory systems there is usually some type of message-passing data synchronization method utilized.

The sharing of the bus results in a number of widely recognized problems. Bus contention when two or more request the bus simultaneously forces the use of a bus arbitration scheme. One processor wins access to the bus, while all others are forced to wait until they are granted exclusive bus access at a later time. Associated with the arbitration is a latency in determining the bus access winner. This can be especially detrimental if a number of processors are involved in a sequence of short data transactions on the bus, and ownership of the bus is for brief intervals only and changes often. The bus arbitration schemes available are numerous, but typically one or two are implemented in a hardware design. This may cause a performance penalty by forcing a particular scheme to be used (i.e. the one that was implemented in hardware) whereupon some other scheme may have been more appropriate. In addition, bus arbitration schemes can be quite complex when special cases are introduced to increase bus performance or utility (e.g. burst mode transfers, locked bus accesses). This can result in both increased implementation complexity and increased arbitration latency. It is stated that for general multiprocessor systems, a global bus oriented system can support about ten processors efficiently, and can be pushed to twenty to thirty [Ston87a]. This limitation arises from the bus contention issue.

The shared bus also suffers from physical limitations. Each processor connected to the bus adds additional load and length to the bus. The increased capacitance and inductance physically degrade the speed of transitions of the electrical signals on the bus, and the increased length produces longer bus settling times. Thus as more processors are added, the data transfer bandwidth of the bus diminishes, and the bus arbitration latency increases. This further compounds the bus contention problem as more and more processors are added.

An advantage of the shared bus system is the conceptual ease of use. Global resources are often memory mapped in each processors local memory map. Accessing this resource is as simple as accessing any other memory location in this address space. The program and data synchronization techniques, such as semaphore management and message passing are widely studied, understood, and implemented. In addition, there exist many commercially available shared bus systems e.g. VME, Multibus, FutureBus.

A method of decreasing the bus contention that is often suggested but seldom used is to incorporate multiple global buses. Each additional bus increases the available bus bandwidth proportionately, thus decreasing the bus contention. Since most global buses are parallel in nature, and commonly up to 32 data and 24 address bits wide (which increase as processor data paths and address ranges increase with time), adding additional identical buses quickly pushes the hardware complexity to beyond manageable levels. The additional buses also add to the complexity of the bus arbitration logic, increasing cost and decreasing speed. A special case of the multiple global bus structure is the crossbar topology, to be discussed separately.

2.3.1.2 Local Bus Topologies

Local bus interconnections are those that share a resource between only two interconnecting processors. The number and pattern of interconnection between processors can vary greatly. Common interconnection patterns are linear, mesh, ring, star, and hypercube, and complete [Haye88a], some of which are illustrated in *Figure 2.2*. Perhaps the most often implemented of these is the ring bus architecture. This is due in part to the increased hardware costs associated with more complex interconnection schemes. However, there is perhaps a growing trend towards these more complex structures, as witnessed by the built in hardware support for six direct-connect communication ports (no hardware glue required) in Texas Instrument's new TMS320C40 DSP processor [Texa90b].

The characteristics of the ring bus will be discussed, but these can easily be extended to other, more complex locally connected architectures. The ring bus displays a high intercommunication bandwidth with its nearest neighbors. The logically short interconnect path leads to physically short interconnections, which simplifies interconnections and results in higher obtainable data transfer speeds due to low bus loading. Since resources in the bus are only shared between two processors, the addressing requirements are generally much smaller than on a shared bus, with a corresponding decrease in implementation complexity of the data path and control logic. Since the bus is less complex overall (generally speaking), it is suitable for scaling with technological advances, i.e. it is easier to apply a new, faster transport medium (e.g. fiber optics) or the latest new IC logic family, and see a corresponding increase in speed.

A disadvantage of the ring bus is the latency in intercommunication between two non-adjacent processors. This arises from the data having to pass through each intervening node to get to its destination. In simple ring bus implementations where the processor is involved in routing the data (i.e. read the incoming data packet, check if it addressed to

me, if so store it locally – else forward it to the next node), the latency can be quite significant. As mentioned previously (section 2.2.2.4 *Computation/Communication (R/C) Ratio*.) the use of an I/O processor can relieve the main processor from intervening in the communication function, and the latency per node can be greatly reduced.

Since local interconnects generally use a distributed-memory architecture, message passing must be implemented to control the synchronization of the data flow [Haye88a]. The previously mentioned barrier synchronization scheme (section 2.2.2.5 *Data Flow Synchronization*.) can be considered a message passing scheme in which all processors involved must inform each other that they are done the current computational cycle and are ready to proceed to the next. A processor will not proceed with further computation until it has received messages from all others that they are done as well. This synchronization can be implemented locally (i.e. like a data flow machine) or at the global level involving all the processors in the system.

Figure 2.2 Local Interconnect Architectures

2.3.1.3 Multilevel Switched-Network Topologies

Multilevel interconnection schemes connect a set of processors to a set of resources through a series of switches arranged in layers. A typical interconnect layer will consist of a set of simpler switches, such as the two-in, two-out switches shown in Figure 2.3. In this case the switch can connect straight through or cross the inputs to alternate outputs (two connections are maintained at all times). There are many arrangements of the interconnect paths between switch layers, but the shuffle-exchange arrangement shown is one of the most popular. The paths can be configured so that any processor can access any memory. As long as no two processors are attempting to utilize the same memory simul-

taneously, there is no bus contention, and the bus bandwidth for each access is high. If contention occurs, one of the processors must wait for the memory to become free. This structure lends itself well to applications that require momentary exclusive access to a shared memory location[*Ston87a*]. The conditions under which this type of access works well are restrictive, and thus it may not work well with some applications that require the sharing of a small amount of memory locations simultaneously. Cases often occur in which the processor computations are synchronized and they are performing the same set of computations, and thus access the memory resources at approximately the same time. It may take careful planning of the sequence of accesses to avoid contention, and may not be possible.

The multilayer network can be expensive to implement due to the complexity of the interconnection, arbitration, and control hardware. There are some commercially available ICs to aid in constructing a multilevel interconnection system [*Texa86a*] by providing the switching network components, but the complex control logic is left to traditional hardware implementation techniques. Additionally, due to the multiple levels of logic that must be switched and data signals must pass through, hardware implementations can be slow.

Figure 2.3 Three Level Shuffle Exchange Multilevel Interconnect

2.3.1.4 Crossbar Interconnection Topologies

The crossbar interconnection scheme can be considered the opposite of the shared bus – it has the least contention and the greatest complexity. For N processors and M memories, there are $N \times M$ switches required and $N + M$ buses. *Figure 2.4* illustrates a 4 by 3 crossbar having 12 switches and 7 buses. The crossbar allows a maximum of $\min\{N, M\}$ simultaneous accesses to the memory resources, as long as they are to different memory modules. Bus contention occurs if two or more processors simultaneously request access to the same memory module. However, memory allocation techniques can be used to spread accesses more evenly over the memory modules. For example in general purpose processing, sequential data elements can be arranged to lie in successive memory modules [Ston87a]. If two processors are accessing the same sequence of data, then after the first access contention one will follow the other in stride through successive modules with no contention. Similar to the multilevel interconnection case, if one or only a few memory locations are shared between many processors, the contention for these can be high. Thus certain algorithms may be unsuitable for this interconnection architecture.

Performance wise, this topology can have a high bandwidth because there is only one switch in any intercommunication path. The implementation cost may be prohibitive due to the large number of switches and data buses inherent in this architecture. Additionally, the arbitration and control logic can be very complex and thus expensive. There is some hardware support available to reduce the implementation complexity, in the form of commercially available ICs [IDT90a][Texa86b].

2.4 DSP Oriented Multiprocessor Hardware Implementations

It is a difficult task reviewing pertinent existing systems, for a number of reasons. There are many systems built by private companies to solve specific internal requirements. These are proprietary to the company involved, and not commonly reported in public literature. Of those reported, there is often not sufficient information published to allow a fair comparison to other systems.

Commercially available systems often have only glossy information published, reporting only the best aspects of the system, while suppressing deficiencies of the approach taken. To obtain more information, it is generally necessary to buy an actual system and benchmark it – not a very practical option. For this reason, only one system in this category will be discussed below.

Figure 2.4 Four Processor, Three Memory Crossbar Structure

Academic research systems are often hypothetical or proposed to be implemented, often relying on simulations of the system for benchmark data. Many are also too narrowly focused in scope to be considered for comparison in our application. If built, often the benchmarks given are very sketchy in details as to what was actually done, rendering them difficult or impossible to interpret for comparison.

What follows is a comparison of those systems that were designed to solve the problem of running general DSP application algorithms. The systems chosen are by no means exhaustive of those available, but represent a number of approaches to the problem, were actually implemented, and had a reasonable amount of information about them published.

2.4.1 Commercially Available Systems: The ASPI Banshee Multiprocessor

The Banshee DSP processing board [Atla90a] by Atlanta Signal Processors Inc. is a TMS320C30 based IBM PC-AT compatible plug-in card. The card is typical of the many commercially available PC plug-in DSP coprocessor boards: 512 Kilobytes of zero wait state RAM, a daughter-board interface for add-on modules (typically for I/O), and communication provisions for data and control with the host PC.

A *MultiProcessor Interface* (MPI) daughter-board [Atla91a] is available to allow a ring bus intercommunication path between multiple Banshee cards in a PC, independent of the PC host bus. It is a shared memory ring structure, with a local 2k by 32 DPRAM residing on each card. The peak data transfer rate is stated as 22.2 Mbytes/S. There is a

common SYNC function provided for system synchronization.

The ring bus is implemented as two separate .left. and .right. bidirectional ports to the two physically adjacent boards as shown in *Figure 2.5*. The bus consists of 32 bits of data and 14 bits of address information, local synchronization signals, and interrupts from messaging. There is also a common (shared) bus portion for clock distribution and a processor synchronization line. One of the boards is delegated as the clock master, and all other boards are configured to use this clock. It is propagated in a daisy chained fashion to all other boards from the master, and is normally used as the C30 CPU clock.

Figure 2.5 The Banshee MPI Block Diagram

The bus structure allows each C30 access to the right side of the local DPRAM, and to the left side of the DPRAM residing on the PE to the right. This interface results logically in a set of processors that are interconnected by an intervening DPRAM. As such, any non-adjacent communication must occur via processor intervention to transfer any data not meant for itself, to the next processor in the proper direction. This process is not transparent to the programmer and must be taken into consideration via software.

2.4.2 Research Systems

In this section, a number of systems that meet some or all of certain criterion will be reviewed. The criterion are that they must have been actually constructed, have meaningful performance results, must be directed primarily at DSP algorithm work, and have architectures that may be applicable to this system.

2.4.2.1 The HPTS

Albert *et al* [Albe85a] at Boeing Aerospace Company designed a parallel processing system for 100 + MFLOPS performance, dubbed the HPTS. Recognizing that memory contention and control overhead are detrimental to the performance of the multi-processor, triple-ported memories and a unique interconnection scheme were utilized. Within such memory array a dual bank switch scheme is used to allow simultaneous processing of one bank while the other is being filled with new data. In total there are 8 processors and 12 memory arrays. The bus structure is a combination of a ring consisting of four processors with a memory array intervening, and a global bus where each processor has access to all of the memory arrays. The remaining four processors are arranged to form a special input processing section that communicates to the ring section via the global bus. At the time the paper was published a prototype was being built, and no performance results were present. It serves as an interesting example, and to highlight that the need for such a system was so great as to spur the development of a working system.

Figure 2.6 HPTS Partial System Block Diagram

2.4.2.2 The PT8

Morgan and Silverman [Morg85a] constructed an eight TMS32010 DSP Processor based system, the PT8. They evaluated the system using DFT and FIR algorithms for application in speech processing. The PT8 consists of eight TMS32010 processors con-

nected as slaves a Motorola 68000 master processor, via a VERSABUS global bus. Four characteristics are listed for their system:

- . the slaves are identical modules which run independently,
- . the slaves cannot pass data directly to each other,
- . the master can broadcast data to any or all slaves and read back individually,
- . conforms to DSP applications where the ratio of input to output data is high.

The maximum data transfer rate between the master and slave(s) is 200 K words/sec.

When performance was tested using a 1K FFT, it was found the PT8 was 3.6 times faster than a single TMS32010. Using a different data flow arrangement, a .PT4. system (using 4 slaves) would be just as fast as the PT8. The limiting factor is the speed of intercommunication between the master and slaves. The computation: transfer ratio was $15\text{mS}:10\text{mS}=1.6$. Two FIR filter examples were tested. The first consisted of eight separate 81 tap FIR filters and the second example was five 96 tap FIR filters. In both cases there was about a 7.5 times increase over a single TM32010.

The conclusions that when the input/output ratio of data is low, then the bus impairs the performance. Another shortfall was the ability to efficiently exchange data between slaves. As well from the data given in the FFT example, the data computation: transfer ratio was low, which emphasizes the effect of the limited intercommunication bandwidth.

2.4.2.3 The Texas Instruments Odyssey

This is a system [Gass87a] based upon a global bus interconnection scheme. There are four TMS32020 DSP processors on each circuit card that fits in a host computer, the Explorer (a LISP based workstation). Each card has a local global bus (the Odyssey bus) and an interface to the host bus (NuBus). Multiple Odyssey cards can be connected to the host backplane. The memory of each DSP is dual ported and one port is mapped onto the global bus (both the local global and host global buses). Thus any DSP can directly access any other DSPs memory. A lock mode is provided for global accesses.

The system is targeted at speech processing, neural simulation, and EEG analysis. Several applications were implemented on the system. Unfortunately no details of performance gains or global bus saturation effects were given.

2.4.2.4 The MUPSI System

This system by G. Bolch *et al* [Bolc87a] consists of custom 'task processors' that are arranged on a unidirectional, registered, ring bus. This is a form of the Pierce Ring, but with a 32 bit wide data and 32 bit wide control information making up one packet. Each task processor contains an independent Data Exchange Unit (DEU) that interfaces the ring bus to the processor. The ring bus itself can be considered as a set of registers connected in a closed ring. The DEU is shown in simplified form in Figure 2.7. The packet transfer rate of the bus is 2.5 MHz through the registers.

Figure 2.7 The MUPSI Data Exchange Unit Block Diagram

This ring bus has the characteristics of unidirectional data communications, a fixed latency of communication between designated processor pairs, and extensibility to more processor without additional hardware complexity. Bolch *et al* claim that in most DSP applications of interest to them, the communications graphs were either linear, or tree-like with the branches consisting of linear sub-graphs. The communications occur mostly between neighboring processors in one direction, and relatively little communication occurs between non-neighboring processors. The application is partitioned into individual parts so that each processor works in a macro-pipelined basis on the data – it uses an incoming sequence of data and produces an output stream of data that goes to the next processor. Each processor performs a relatively large subtask. The pipelined bus latency could be ignored for applications with linear communication graphs, but in applications with parallel paths the path latencies had to be balanced. Applications with closed loops

had to ensure the added latency did not exceed the allowable loop delay.

Results were given for a vestigial sideband data transmission system, which has a completely linear data flow graph. The results (table 2 of [Bolc87a]) are reproduced here in Table 2.1.

Table 2.1: Vestigial Sideband Example Execution Results

No. of Processors	1	2	3	4
Processor 1 Loading	220	116	84	65
Processor 2 Loading	n/a	116	76	64
Processor 3 Loading	n/a	n/a	84	55
Processor 4 Loading	n/a	n/a	n/a	72
Min. Cycles/Sample/Proc ($208/N+12$)	220	116	81.3	64
Max theoretical speedup	1.0	1.89	2.68	3.43
Actual Speedup	1.0	1.89	2.62	3.06

The task consists of 208 cycles/sample plus 12 cycles of task independent overhead (interrupt is given by return, etc.). Therefore the minimum run time is given by $(208/N+12)$. For a four processor system where all processors can be loaded fully and there is no intercommunication time, the number of cycles to compute the result for each sample is 64. This is a speedup of 3.43; the basis of the calculation of the maximum theoretical speedup. The difference between this and the actual observed speedup is due to non-uniform processing loads because of indivisible computational blocks. The effects of processor intercommunication were not mentioned in the results, implying that the latency effects were negligible with respect to the computation length and unbalanced processor loading.

2.4.2.5 The DSP.* Multiprocessor

This DSP-32 DSP IC based multi-processor system [Kahr88a] was designed for high quality digital audio mixing, of this architecture is the use of a commercially data from one processor to another in a pseudo static manner (i.e. can be changed on the fly but at a rate much slower than the intercommunication data speed). The incoming serial data

from the processors is 'connected' to serial inputs of another processor. The serial inputs are time division multiplexed and the switch operates as a time slot interchange crossbar, directed inputs at a certain time slot to the predefined outputs. A commercially available telephone switch was used to perform the serial time-slot interchange crossbar function. The serial data rates are 705.6 K bits/S, allowing up to 12 inputs to be switched to a processor in one sample time from other processor's outputs. Although intended for digital audio mixing, the concept of quasi-statically varying unidirectional data paths is applicable to many DSP application data flow graphs.

2.4.2.6 The SMART Architecture

P. Hoang *et al* [Koh88a][Hoan90a] at the University of California describe a multiprocessor system designed to speed up simulation of DSP behavioral algorithms by two orders of magnitude. Each processor is based on a DSP32C DSP IC. Intercommunication between processors is accomplished via a pseudo-statically reconfigurable bus to match the communication pattern of the algorithm. The bus or portions thereof can be configured via software control to match pipelined or parallel concurrency. Thus portions of the bus can operate in a pipelined manner while others can operate in a shared memory (global bus) fashion. Figure 2.8 illustrates three processing elements.

Figure 2.8 The SMART Multiprocessor Block Diagram

The data flow graphs considered in the paper are linear data flow graphs with pipelined and parallel unidirectional data paths. The switch S is connected to the bypass

unit for pipelined portions and the memory for parallel parts. An important feature is the bypass unit, which allows a processor to send data to a non-adjacent processor within a pipelined section without involving the intervening processors.

Others architectural features are a broadcast data distribution mechanism, and a barrier program synchronization mechanism. The claim is made that barrier synchronization is adequate for most cases of synchronization in DSP applications.

In [Koh88a], benchmark simulations were performed of various DSP algorithms mapped onto the architecture in different ways. Table 2.2 restates these results in a slightly modified manner. These modifications concern condensation of the data for clarity of presentation (the original tables are eight in number). In many cases the median value range is presented with outlier values ignored, so as to present a better feel for what the majority of processors are doing. This may be considered subjective in nature, and [Koh88a] should be consulted for more precise results. In the cases of configured buses (i.e. those that have some sections pipelined and others parallel so as to match the algorithm data flow), there are two sets of values given that represent the pipelined and parallel sections separately. Most of the table headings are self-explanatory. The bus use is the percentage of time that the bus is actually being used. The bus congestion is the number of processors that request the bus when the bus is being used. By definition this value is 1.0 for the pipelined case. The read/write value is the amount of time the processor spends reading and writing data.

Section 4.1 of this paper compares an FFT partitioned onto 10 processors with fully pipelined and fully parallel bus configurations. Obviously the shared bus is saturated, but the bus is not being utilized 100 percent of the time (only 73%) because is congested only during certain intervals of the processing, as indicated by the higher bus congestion number. It is also noted that the FFT is a computation bound problem, as a relatively small amount of time is spent intercommunicating data (approximately, 30%).

Table 2.2: SMART System DSP Application Simulations

Algorithm	Bus Config.	No. Proc	Speedup per Proc.	Proc Idle Time (%)	Bus Use (%)	Bus Congestion	Read/Write (%)
FFT	Pipelined	10	.796	6-20 (most 20)	15.15	1.0	36
	Shared	10	.497	0-50 (most 40-50)	73	3.92	39
Echo Cancellor	Configured	11	.772	1-5 parallel	< 6 pipelined 1.3 parallel	1.0 pipelined 2.63 parallel	3-5 pipelined 3-11 parallel

Table 2.2: SMART System DSP Application Simulations

Algorithm	Bus Config.	No. Proc	Speedup per Proc.	Proc Idle Time (%)	Bus Use (%)	Bus Congestion	Read/Write (%)
Pitch Extractor	Configured	15	.807	20-30 pipelined 0 parallel	-7.5 pipelined -.3 parallel	1.0 pipelined 3.1 parallel	-20 pipelined -2 parallel
	Configured	61	.816	15-20 pipelined 20 parallel	-40 pipelined -4 parallel	1.0-1.44 pipelined 18.74 parallel	-34 pipelined -1.8 parallel
	Shared	61	.336	0-66	85.6	29	1.8 - 33
Matrix Multiply	Pipelined	16	.575	2-36	38.6	1.0	-15
	Shared	16	.419	5	27.6	8.72	-15

The echo canceller simulation contains both parallel and pipelined data paths. It is claimed that the adaptive portions of the algorithm cannot be pipelined because they contain directed feedback loops, and thus must be parallel paths. This assumption serves well for demonstration purposes, but other research indicates that delay of several sample periods can be tolerated in practice, with negligible effects on the performance of the adaptation dynamic and steady-state response [Darl88a] [Haim90a] [Kaba83a] [Long87a]. Perhaps if the echo canceller was fully pipelined using delayed coefficient update, then the efficiency (speedup per processor) might be better.

The pitch extractor is an example often used in multiprocessor literature for demonstration purposes. Here the efficiency per processor varies little (approximately 0.8) with the number of processors in the system (from 16 to 61). This indicates that the bus structure is very efficient for this problem. When configured for a shared bus, the effects of global bus contention are clearly illustrated. In addition, the increase from 15 to 61 processors shows that the problem transitions from a computationally to intercommunication bound problem, as witnessed by the increased percentage of bus use and data intercommunication time. Not shown in the table is data for the fully pipelined bus configuration for the pitch extractor. It is claimed that the performance was almost as good as for the optimally configured bus.

A later paper on the SMART system presents results of actual algorithm partitioning onto a 10 processor system. This paper is mainly concerned with the automatic partitioning of the algorithm onto the system, and the results may not reflect the optimal partitioning performed manually for the previous simulation results. They are however

indicative of what can be achieved using this architecture. These results are reproduced in, as well as the previous predictions obtained by simulation where applicable. It is assumed that the bus is configured to suit the algorithm by the partitioning software. It is unknown why only eight of the ten processors available were utilized.

Table 2.3: SMART System Actual Results

Algorithm	Previous Simulation		Actual System	
	Speedup/Num. Processors	Speedup per Processor	Speedup/Num. Processors	Speedup per Processor
Pitch Extraction	12.1 / 15	.807	7.07 / 8	.884
DFT (pipelined)	7.96 / 10	.796	7.53 / 8	.941
Matrix Multipliy	9.2 / 16	.575	8.0 / 8	1.0
Cordic	n/a	n/a	6.97 / 8	.871
Equalizer	n/a	n/a	6.45 / 8	.806
Dynamic Time Warp	n/a	n/a	8.0 / 8	1.0

2.4.2.7 The RAP Multiprocessor for ANN Simulation

Morgan *et al* [Morg90a] have designed and built a ring based multiprocessor for the simulation of artificial neural networks (ANNs). The ring Array Processor (RAP) consists of multiple 4 processor boards connected in a registered ring topology, with a 68020 based host processor, all on a VME backplane.

The structure of a single processing node bus interface is shown in *Figure 2.9*. The structure was chosen from the ANN algorithm intercommunication requirements. The ANN weight sum calculation is divided across all processors in the system, and each partial weight sum must be broadcast to all other processors in the system with efficiency. With a circular pipeline interconnection bus, the partial weights are written by each processor once onto the bus, and then are shifted along the bus. As each partial weight passes by each processor, it is read and accumulated to form the total weight value. Simultaneously, it is shifted along to the next processor. Each processor must write once and read $N-1$ times to accumulate the total weight, where N is the total number of processors in the system. The bus is synchronous to the processor clock, so that there is one shift per pro-

Figure 2.9 Bus Interface Structure of a RAP Processing Node

cessor cycle (62.5 nS or 16 MHz). This is equivalent to a 205 Mwords/S data transfer rate (820 Mbyte/S).

2.4.2.8 The Berkeley/Dolby System

Bier *et al* at the University of California, Berkeley [Bier90a] have used a commercial DSP multiprocessor system with their .Gabriel. data flow programming environment for DSP algorithms. The commercially available Dolby system consists of four Motorola DSP56000 processors, each having a local memory and one global shared memory. The static scheduling of Gabriel ensures no more than one processor at a time reading or writing to global memory location, thus the overhead of indivisible test-and-set operations is not required. The only run-time overhead incurred is for bus contention resolution and semaphore handling. The minimum length of a shared memory transaction is 30 instruction cycles, but in a telephone channel simulator example, the transactions took up to 40 cycles to complete due to bus contention. In this example, it is noted that each processor spends approximately 30% of its time on interprocessor communication.

The paper goes on to propose some architectural variations to reduce the overheads that were observed. A hardware .gatekeeper. could be used to perform the semaphore management functions. The hardware can perform the same functions as the software, only much faster. Another proposal is to incorporate a central bus controller that grants the bus in a pre-specified order based on the order determined a priori by a static

scheduler. This scheme has the advantage of not requiring any explicit hardware or software to perform bus contention resolution or semaphore management, and if the scheduler can do a good job with the algorithm, then a very high shared memory access bandwidth can be achieved. Simultaneous have indicated that an eight times speedup can be achieved over the Dolby architecture. The static scheduling success depends on the synchronous data flow paradigm being applicable to the problem. Another variation proposed that is under evaluation, is to eliminate the global memory in favor of a FIFO located on each processor. When the sender of data is granted access to the global bus, it simply writes the data to the appropriate processors FIFO. It is worthy of note that this method is inherently unidirectional in nature.

2.5 Conclusions

A review of the DSP algorithm data flow and partitioning literature indicates that many DSP algorithms are suitable to implementation in a linear, pipelined fashion. When considered in this way, the intercommunication between subblocks of the algorithms of interest generally exhibit the following characteristics:

- . data flow is synchronous.
- . unidirectional data flow.
- . data flow latency effects can be tolerated.
- . there is locality of communication between PEs.
- . the effect of non-local feedback loops can be minimized.
- . data processing method is periodic and is performed both on a sample and block basis.
- . the temporal distribution of the intercommunication is generally fixed a priori and occurs in bursts which can result in data traffic congestion.
- . the algorithm computational complexity versus communication requirements (R/C ratio) tends towards large-grain.
- . data flow synchronization can generally be controlled by barrier synchronization.

Some common multiprocessor interconnection architectures were reviewed, and their advantages and disadvantages were discussed. No effort was made at this stage to correlate these interconnection methods to the algorithm characteristics, as this will be undertaken in chapter 4.

A brief review was done on proposed and (more importantly) implemented multi

processor systems that were designed to run DSP-type algorithms. The details of these systems will be considered again in chapter 4 when our intercommunication structure is chosen.

Chapter 3 Candidate Algorithms

3.1 Introduction

Some algorithms of research interest will be examined in light of the previously discussed data flow characteristics. Two such communication system algorithms are the *High Data-rate Subscriber Loop* (HDSL) receivers, and the MEL-Cepstrum speech processing front end. In addition, due to the typically long length of the *finite impulse response* (FIR) and *adaptive FIR* (AFIR) filters in many systems, these will also be briefly examined in terms of partitioning for concurrent implementation. The two system level algorithms were also used as benchmark examples for testing the multiprocessor system, to be reported in chapter 6. These examples were chosen as typical of research interests ongoing at Carleton University.

3.2 DSP Application Algorithm Examples

Three DSP algorithm examples will be considered as a basis for illustrating requirements the system hardware design attempts to satisfy. The examples can be roughly categorized into two parallelism task granularity paradigms: fine grain and large grain. A finite impulse response (FIR) filter with either fixed or adaptive coefficients would be considered of finer-grain granularity. This is because it is a fundamental DSP building block (or subblock) that will be partitioned into smaller parallel computational units with a corresponding increase in the R/C ratio (see section 2.2.2.4 *Computation/Communication (R/C) Ratio* on page 11). By reviewing the data flow characteristics of a simple DSP subblock such as the FIR filter, useful verification of the data flow characteristics previously discussed can be obtained. The intra-block partitioning of the FIR may be of use in the system algorithms due to the long lengths of the FIR and adaptive FIR filters encountered. Intra-block partitioning these long filters can be performed to achieve computational load balancing. The two DSP system algorithms, an ISDN/HDSL trans-

mitter/DFE receiver, and a MEL-Cepstrum speech processing front end, can be classified under larger-grain granularity: there is more computation performed per unit intercommunication. This classification is not rigorous, but serves to delineate the examples into different categories.

3.2.1 FIR Intra-algorithm Data Flow

Figure 3.1 displays the data flow for a FIR filter. The input data is processed as an 'infinite' input vector i.e. one data sample is consumed every processing step. *Figure 3.2* and *Figure 3.3* illustrate two possible methods of partitioning the FIR over three PEs:

- . in a pipelined manner
- . in a parallel manner

Latency of the final data output is introduced within each PE (retiming delays) to align the partial sum calculation so the correct pieces are correctly accumulated. Notice the implied delays explicitly shown between PEs due to inter-communication (intercommunication and retiming delays are shaded). The partitioning can also be arranged in a combined pipelined/parallel manner if so desired, but this is not shown here.

Figure 3.1 FIR Filter Data Flow Diagram

The pipelined method of partitioning seems more natural, as it more evenly distributes the computational loads. The parallel partitioning leaves the accumulation of the partial sums as a somewhat unbalanced load. However this effect would be minimized in a practical FIR example where there are a large number of taps and the final partial sum calculation is a small percentage of the total calculation. The parallel partitioning also generates the

Figure 3.2 Pipelined Partitioning of the FIR Filter

Figure 3.3 Parallel Partitioning of the FIR Filter

need to simultaneously broadcast the data to all PEs from the input source. If a broadcast data distribution mode is not provided by the hardware, then $M-1$ non-adjacent bus transactions will be required (with M PEs in the system), as opposed to $M-1$ local bus transactions in the pipeline partitioned case.

For the pipelined case, the required retiming delays for the partial sum alignment can be shifted from the partial sum output to the PE input as shown in Figure 3.4. This creates a more regular algorithm structure in each PE, which can simplify programming of the algorithm onto the system. Here the PE intercommunication latency is designated by

the shaded delay block.

Figure 3.4 Retimed Pipeline Partitioned FIR Filter

A simple activity chart for each PE is shown in *Figure 3.5*. Since all PEs are performing the same task, and the start of the computation of each PE is assumed to be synchronized to computation on all other PEs, it is not surprising to note that the intercommunication takes place at approximately the same time. This is shown in the intercommunication volume graph in the lower part of the same figure.

From the retimed FIR data flow diagram, within each PE we can observe:

- . the data communication is unidirectional within each PE.
- . the flow of data computation and communication is deterministic i.e. there are no data dependant flow decisions
- . the data consumption/production is grouped in bursts in the time domain i.e. the input data is consumed at the start of processing and the result is produced at the end of processing (the latter is a general statement – in reality the data consumption/production time can be shifted somewhat by the use of programming techniques, but the time of occurrence is still invariant with respect to the start of the processing)

Additionally from the data flow and intercommunication activity graphing between the PEs we can observe:

- . the data communication is unidirectional between the PEs.

Figure 3.5 FIR Activity Chart and Communication Volume

- . all intercommunication occurs between adjacent PEs.
- . if the algorithms are started on all PEs at approximately the same time (due to program synchronization), the communications between processors will occur at approximately the same time i.e. the intercommunication occurs in bursts.

This rather simplistic view of the FIR subblock is used to confirm that its data flow characteristics are similar to those reviewed in chapter 2, thus building confidence in the understanding of data flow characteristics for DSP algorithms of interest. It is felt that this can be extended to real-world applications which are of the same structure, varying mainly in the size of the filter.

3.2.2 The HDSL Transmitter/Channel/DFE Receiver

The high-speed data subscriber loop (HDSL) is a 800 bit per second data transmission system over domestic phone lines up to 10 K feet in length. The proposed system transmits 2 bits per symbol as a four level quaternary amplitude signal (2B1Q) at a 400 K symbol/second rate. For research purposes, performance evaluation is of interest, including bit error rates vs. channel noise and crosstalk. This example is based on thesis work by P. Mohanraj [Moha90a].

The 2.5 μ S symbol period is short relative to currently available DSP processor instruction cycle times (at approximately a 60 nS CPU cycle time, 41.67 instructions can be executed). The HDSL system as illustrated in Figure 3.6 consists of long FIR and

adaptive FIR filters (on the order of 300 – 400 taps). To simulate one symbol period requires approximately 1800 multiplies and 2900 additions¹, as well as control overhead. This is beyond the capacity of a single DSP processor, and on fast general purpose workstations the simulation times are prohibitively long. For example, simulation at Carleton University by P. Mohanraj [Moha90a] for 3000 transmitted symbols exceeded 20 hours on a SUN SPARC I workstation. To simulate the few million symbols required to evaluate bit error rates on the order of 10^{-6} is not feasible.

Figure 3.6 HDSL Transmitter, Channel Simulator, and DFE Receiver

The data flow of the system is pipelined, or can be made pipelined by simple retiming, as shown in Figure 3.7 for the transmitter/channel simulation portion of the sys-

1. These values were calculated as follows, using information on the number of taps required for a typical HDSL simulation from [Moha90a] and FIR/AFIR assembly language programming information from chapter 12 of [Texa90b] (FIR takes 1 multiply, 1 addition; AFIR takes 2 multiplies, 3 additions).

FIR Source	Number of taps	Type	Multiplies	Adds
Channel	400	FIR	400	400
Crosstalk	25	FIR	25	25
Echo	300	FIR	300	300
Noise	32	FIR	32	32
Echo Cancellation	300	AFIR	600	900
Feed Forward Equalization	21	AFIR	42	63
DFE	200	AFIR	400	600
Totals:			1809	2920

tem. The retiming delay blocks (.D.) are the pipeline delays that are required when the data flow between subblocks occurs between physically separate PEs. If two subblocks are resident on the same physical PE, then no retiming delay is required. In this example, the subblocks can be partitioned onto up to four separate PEs. Since each subblock generates random data in this example, they can be grouped together in any order (without the retiming delay between grouped subblocks) to help balance the computational loading in the system.

During each symbol period, each block consumes a small amount of data at the beginning of the computational cycle, and produces a small piece of data at the end to be used by the next block in the next computational cycle. Since the majority of the subblocks are either FIR or AFIR filters, the timing of the data traffic is very similar to that shown previously for the FIR filter. The HDSL system can simplistically considered a single, very long FIR filter with different sections having different sets of coefficients and different coefficient update rates. This analogy indicates the volume of data exchange occurs in short high volume bursts at the beginning and end of the cycle, assuming that the fir partitioning is such that all PEs have roughly the same computational time. In terms of partitioning the system on single subblock boundaries, this may only be partially valid, but in the context of load balancing on a multiprocessor system this becomes more true. The blocks are grouped in such a way that the computational periods on each processing element (PE) are as equivalent as possible, therefore interprocessor communication would occur closer to the period boundaries.

Figure 3.7 Pipelined HDSL Data Transmitter and Channel Simulator

The HDSL algorithm can also be easily divided in the temporal domain into sym-

bol periods. The symbol periods are the natural way to partition the algorithm into infinitely recurring computational cycles. This suits the barrier synchronization model well.

3.2.3 MEL-Cepstrum Speech Pre-processing

The Mel-based cepstral coefficients are a representation of speech signals that are used to characterize the speech signal more efficiently. It can be used as the front-end of a speech recognition system, to be followed by some sort of feature extraction algorithm to recognize patterns in the coefficients. The relative merits of this representation will not be discussed here, but can be found in literature [OSha90a][Pars87a]. The processing steps involved in the algorithm are illustrated in the data flow diagram in *Figure 3.8*. The speech input vector is partitioned into overlapping blocks of input data and then transformed into the frequency domain via an FFT. The frequency values are non-linearly

Figure 3.8 Mel-Cepstrum Algorithm Data Flow Diagram

(frequency-warped) based on triangular filters whose center frequency and bandwidth follow a .mel-scale. i.e. the center frequencies become more widely spaced and the bandwidth widens as frequency rises. The resulting magnitudes are then warped according to the logarithmic scale. The first 8 to 14 coefficients of the inverse FFT of these values are then computed, using a simplified transform.

The algorithm manipulates a relatively long block of input data (e.g. 256 or 512 speech samples) into a smaller set of 8-14 representative coefficients. At each subblock of the algorithm, a block of data is consumed near the beginning of the computational cycle and a processed block of data is produced at the end. Each subblock varies widely in its computational complexity. The initial FFT operates on the largest block, after which the block size is reduced by the mel-scale filter, thus the FFT is expected to be the longest computational block. This in turn implies that computational load balancing might be a problem. However, the standard FFT can be highly optimized on DSP processors, would help in load balancing.

3.3 Conclusions

Algorithms that were felt to be of future interest were reviewed. Their inter-block data flow was briefly examined to ensure that the data flow generally adheres to the previously stated data flow characteristics of DSP oriented systems. The algorithm overview also serves to introduce examples that were used to benchmark the system performance, as described in chapter 6.

PAGINATION ERROR.

ERREUR DE PAGINATION.

TEXT COMPLETE.

LE TEXTE EST COMPLET.

NATIONAL LIBRARY OF CANADA.

BIBLIOTHEQUE NATIONALE DU CANADA.

CANADIAN THESES SERVICE.

SERVICE DES THESES CANADIENNES.

Chapter 4 Proposed Interconnection Architecture

4.1 Introduction

We have established the system intercommunication requirements in the previous chapters. For clarity we will briefly reiterate the characteristics of the inter-block data flow:

- intercommunication occurs at periodic, deterministic temporal points in the algorithm, with bursts of traffic at the temporal breaks.
- unidirectional data flow.
- loosely observe principle of locality between program partitions.
- program temporal partitioning fits the barrier synchronization model.

From these we can define an intercommunication bus structure. First we will briefly review available bus structures in the context of this application, and then discuss the chosen approach and attempt to solve any deficiencies (as all approaches have advantages and disadvantages). The chosen architecture is then compared to existing systems that were previously reviewed. The intercommunication bus structure proposed here will be further described in detail in chapter 5.

4.2 Bus Architecture Comparison

A concise comparison of bus architectures that could be used in this application will be given. The configurations considered are global bus interconnects, local bus interconnects, and multiple versions of these.

4.2.1 Global Bus Topologies

The global bus requires that all processors intercommunicate over a global pathway. The communication requirements were found to occur in bursts, i.e. at the same point in time. With a global bus this results in contention and its associated delays. Even without contention, arbitration and acknowledgment delays, and physical bus loading effects add significant overhead to each data transfer. Block transfer methods and protocols attempt to minimize overhead for consecutive transfers of a block of data, but at the expense of higher hardware/software cost and complexity. The advantage of block transfers is minimal in many of the applications of interest, where the transfer requirements are small in volume and sporadic. Multiple global bus structures (e.g. crossbar) attempt to overcome the former deficiency of bus contention, but their high hardware cost and complexity preclude their use in many applications, including this one. The high hardware cost is due to the high pin counts for interconnecting processing elements, heavy bus driving circuits due to the high bus loading, and complex control circuits for bus arbitration.

4.2.2 Ring Bus Topologies

The ring bus fits many of the established intercommunication criterion. It has a high peak adjacent processor communication bandwidth due to non-interference with any other processor's concurrent communication. The short physical bus lengths with light electrical loading permit a higher speed design to be implemented. This suits the burst-like nature of the intercommunication traffic which requires high peak transfer rates to eliminate the bottle neck, and the principle of locality which demands the most bandwidth between adjacent processors.

The ring bus is in many cases implemented unidirectionally to simplify hardware design. This is acceptable in our application due to the unidirectional nature of intercommunication traffic. One major disadvantage usually associated with ring buses is the latency introduced when communicating between non-adjacent PEs (in the bus data flow direction). It should be noted a large part of this latency is due to processor intervention to check if the data packet is addressed to itself or not, and then passing it on to the next processor in the ring if so required. Some type of I/O processor hardware would have to be added to make the ring bus viable for the application.

4.2.3 Other Bus Topologies

Other, more esoteric bus solutions were not considered practical for this applica-

tion. Generally relying on a large number of buses from each PE and possibly an active backplane results in unwieldy hardware designs that were beyond the resources of this project to develop. The multiple local interconnects (i.e. star, hypercube), multiple global, crossbar, and multiple level interconnect buses (i.e. shuffle networks) are examples of this approach.

Before dismissing multiple local interconnects entirely, it is interesting to compare a simple unidirectional ring bus with a more fully connected ring bus architecture. The major assumption is that there is a fixed amount of hardware resources available to use. Basically this implies that to add a second bus for a two bus system interconnect, the data path must be one-half of the single bus width. Thus the peak transfer rate of the two buses combined is the same as the single bus. Since there are more interconnection paths, the average latency of communication is reduced, but the speed of the individual links is reduced. As an example consider the five PE single ring bus in *Figure 4.1 a)* and the five PE chordal ring structure in *Figure 4.1 b)*. The period of intercommunication for adjacent PEs on the ring is T_d , and the latency around the ring is L for each node. A further assumption is made that an I/O processor on each PE can reduce the latency for each node to one data transfer time T_d . Since the hardware resources are fixed, and the chordal structure has twice as many interconnects, then they are one-half the width and thus one-half the bandwidth of the ring example (i.e. adjacent processor communication takes $2 * T_d$ amount of time). *Table 4.1* states the intercommunication time for communication from PE 1 to all other PEs for the ring and chordal structures. Contrary to the expected intuitive results, the chordal ring communicates at less than or equal to the speed of the single ring bus, under the assumptions stated.

So far only the case where a single communication between two PEs has been looked at. If there are two simultaneous intercommunication paths to a single PE, then bus contention must be looked at. For example, if PE1 and PE2 simultaneously send data to PE3, then contention occurs on the PE2-PE3 communication link for the single ring bus case if the data from PE1 arrives at PE2 at the same time PE2 would like to send its data. The total intercommunication time will be $3 * T_d$ due to a waiting period for one of the data packets. For the chordal ring, there are two communication paths simultaneously transferring the data in a time of $2 * T_d$. The single ring introduces additional latency when non-adjacent PE simultaneous intercommunications occurs. However this may not be a great disadvantage, since previously it was noted that for applications of interest, the data communication occurs most often between adjacent PEs. The case of communication between non-adjacent PEs that may cause communication interference occurs less often.

Figure 4.1 Single and Chordal Five PE Ring Buses

Table 4.1: Intercommunication Time Comparison for Single and Chordal Ring Buses

Communication Path	Basic Ring ¹	Chordal Ring ²
PE1 to PE2	T_d	$2 \cdot T_d$
PE1 to PE3	$T_d + L = 2 \cdot T_d$	$2 \cdot T_d + 0 \cdot L = 2 \cdot T_d$
PE1 to PE4	$T_d + 2 \cdot L = 3 \cdot T_d$	$2 \cdot T_d + 1 \cdot L = 4 \cdot T_d$
PE1 to PE5	$T_d + 3 \cdot L = 4 \cdot T_d$	$2 \cdot T_d + 1 \cdot L = 4 \cdot T_d$
Note 1: Latency L for the single ring $= T_d$ Note 2: Latency for the chordal ring $= 2 \cdot T_d$		

4.2.4 The Selected Bus Architecture

The ring bus seems to be the most suitable bus architecture because it suits the application data flow characteristics and is practical to implement in hardware. The main disadvantage is the latency introduced by processor intervention when data is passed between non-adjacent PEs. The processor must check the incoming data to determine if it is for itself, and if not then pass it on (data routing). This process consumes valuable processor computational resources that are required if the algorithm simulations are to be per-

formed at or near real-time. The latency introduced can be especially detrimental if overall feedback loops are present in the application. Thus an I/O processor must be added to off-load the data intercommunication routing task from the main processor. Ideally the I/O processor would allow other PEs to be mapped in the address space of the main processor, so that writing to them appears and takes the same amount of time as writing to local memory.

To solve this problem to a great extent, this system will utilize a variant of the ring architecture, coined here as the "slotted-ring". Essentially a separate Input-Output Processor (IOP) is introduced to handle intercommunication between PEs in a pipelined "registered" ring structure, similar to the concept used in the MUPSI [Bole87a], SMART [Koh88a], and RAP [Morg90a] systems. A description of the system concept will be presented, and a comparison to existing systems will be made.

4.3 Slotted Ring Bus Description

Each PE incorporates an I/O processor that operates asynchronously to the main processor. The processor writes to other processors in a memory-mapped address space, and reads incoming data from other processors from a local dual-port memory (DPRAM). The transport of data via the ring bus is totally transparent to the processor. This concept is illustrated in *Figure 4.2*.

Figure 4.2 Slotted Ring Structure for Three Processing Elements

The term "slotted" arises from the presence of a clocked register in each PE's IOP. The IOPs of each PE are clocked by a master bus clock. This behaves as a circular shift

register with parallel data packets. The slots are periods of the bus clock. An IOP may write a data packet into a time slot to be shifted down the bus to be read by other IOP's. The packet contains an address to allow other IOP's to determine if the data should be read from the bus and stored locally for access by the IOP of the processor element. Since the data on the bus has an associated address, it will be referred to as a packet. Thus each packet consists of 32 bits of data and 13 bits of address for a total 45 bits (to be discussed in detail in chapter 5). The mechanism for setting up a protocol to allow orderly passage of data is programmable at run time, allowing flexibility. The structure of the IOP is shown in *Figure 4.3*. The processor can transfer data to another processor by writing to the memory-mapped address block in the processors local address space. The packet passes through a FIFO due to the asynchronous operating of the processor and bus, and is transported onto the bus by the IOP. This is transparent to the processor. The address of the packet is monitored by each IOP as the packet is shifted along the bus. If the address is for that PE or is a special broadcast address, then the IOP takes a copy of the data and places it into the dual port RAM at the address specified by the packet. Thus the packet address can be divided into two fields

- PE address specifying a destination PE(s). The destination can be all PE's (i.e. a broadcast mechanism).
- the DPRAM memory address within the destination PE.

The division between the two fields is programmable by the user.

The method of address comparison is based on a user programmable "tag" value. It is stored in an address tag RAM locally on each PE as part of the local bus controller. The address compare portion is shown in *Figure 4.4*. There is a unique tag location for each bus slot, thus it can have a different value on each time slot. This tag is compared to the incoming bus address most significant bits, and a write from the bus into the DPRAM is initiated when a match occurs. In the simplest configuration, all tag values are the same, and each PE uses a unique tag value e.g. the tag value can be the physical PE number in the system. In this way each PE's local DPRAM appears mapped to a unique address of all the other PE's address spaces. For more complex schemes, the intercommunication can be "scheduled" so that a PE can write the data to multiple predetermined PEs at a time.

This slotted bus overcomes the major disadvantage of the "standard" ring bus - latency. This is because there is no processor intervention in moving a data packet from its source to destination - the IOP performs this transparently. The short interconnect path allows the bus clock to operate at low periods, which further reduces latency. For an N processor system the maximum bus transport latency is N-1 bus clock periods. This is fur-

Figure 4.3 Bus Input/Output Processor (IOP) Block Diagram

Figure 4.4 Address Tag Comparison Block Diagram

ther discussed in section 5.5.5 "Bus Performance".

4.3.1 Barrier Synchronization

The basic concept used here is execution barrier synchronization, that is, a program suspends its execution at some point in its operation (at the user inserted barrier) and

waits for some event to occur before execution can proceed. This mechanism is intuitive in its application to DSP algorithms which typically exhibit natural processing boundaries, usually in time (e.g. speech frames or baud intervals). A barrier is inserted at this boundary in all software modules running on the system, thus starting each computational cycle on each PE at the same time i.e. all software module execution start times are aligned temporally. There are three hardware mechanisms provided in the hardware design that can be used singly or in combination to implement these execution barriers. These are bus interrupts, bus external flag, and signalling bus. They will be discussed in detail in chapter 5.

Figure 4.5 Barrier Synchronization Timing Example

4.3.2 Comparison to Existing Systems

The SMART system [Koh88a][Hoan90a] uses a statically reconfigurable bus to allow both parallel and pipelined bus sections, to suit the data flow of each part of the algorithm. The physical speed of the bus is not discussed by Koh *et al.*, but undoubtedly the increased hardware complexity of this scheme results in a slower data transfer time than a simpler implementation. It is their hope that the better fit of the data flow with the bus architecture will more than offset the decreased speed. In the results of simulations published in [Koh88a] (see Table 2.2, ““SMART” System DSP Application Simulations,” on page 26) they do not directly compare the optimally configured bus to the fully pipelined bus configuration, but simply state that “the performance was almost as good” in one case. This omission is curious, and leads to the conclusion that perhaps the extra bus complexity was not worth the effort.

The bus data exchange unit of the MUPSI system [Bolc87a] is very similar to the

slotted ring. The major differences are in the use of two stages of registers in each nodes data path, the use of a target processor list for addressing, and hardware support for switched dual-buffered RAM for data storage. The dual registers are probably used to simplify design timing considerations, and if eliminated could result in a data transfer rate increase. By pipelining the bus IOP control logic in our design, this redundant register was not required. The resulting data transfer rate due to this change and the use of newer technology is 33 Mwords/S, as opposed to MUPSI's 2.5 Mwords/S. The MUPSI "list of target processors" addressing is replaced by our "tag address matching" system, which performs a similar function. The switchable dual-buffered memory is replaced with a true dual-ported RAM, in which switchable buffers can be simulated with a minimum of processor overhead. The system presented here is in many ways an extension of the MUPSI system, but using technological advances and architectural differences for a higher level of bus performance.

The RAP system also uses a dual-registered ring bus structure, to allow the main processor access to the bus. There is no I/O processor and a simpler synchronized global broadcast mechanism is implemented. The use of the main processor for intercommunication was precluded in our system due to the computational requirements, but seems well suited to the large-grain neural net problems with rigidly defined intercommunications such as it was built to solve. It serves as another example of a similar approach to our system.

The system proposed is felt to be superior to the existing commercial system reviewed (Banshee and MPI, see section 2.4.1 "*Commercially Available Systems: The ASPI "Banshee" Multiprocessor*"). The proposed bus operating speed of 33.3 Mega Words/S is significantly faster than MPI's 22.2 MegaBytes/S (5.5 Mega Words/S). The IOP in the proposed design allows user transparent intercommunication to any other processor in the system, without any extra computational overhead required from the main CPU. In the Banshee system, non-adjacent data transfer requires manual programming effort and intervention by the C30 CPU. The synchronization mechanisms provided by both systems are quite similar.

It should also be noted that the proposed design allows for continued system upgrading since its performance is not related to any particular "short lived" feature of the DSP processor considered.

4.3.3 Additional Hardware Requirements

Further requirements for the hardware system will be stated. As DSP systems grow

in complexity, more processing power is required. A fast general purpose DSP processor must be chosen. It must be flexible enough to cover a wide range of algorithms, as the end use cannot be specifically foreseen. The bus interface should be well decoupled from the main processor so that upgrading either portion independently to a new technology can be accomplished if so desired.

The use of DPRAM in the bus interface affords the use of the message passing location that are built in to most DPRAM ICs. This could be done by distributing the bus side DPRAM message interrupt to all other PEs in the system. Additionally, some type of interrupt bus above and beyond the C30 interrupts would be an asset for future flexibility and expandability.

The multiprocessor system as a whole must have a modular design, to allow expansion by replication. The host interface must require a minimum of hardware and software development, to reduce the scope of the work required. It must still be flexible enough to allow up/down loading of data and programs to all PE's in the system, allow control over the whole multiprocessor system, and control over individual PEs for debugging purposes (i.e. single- stepping, setting breakpoints). As well, there should be an existing library of DSP software routines available to help reduce software development time.

4.4 Conclusion

A variety of intercommunication architectures were reviewed in light of the previously established algorithm data flow considerations. The review included both general bus topologies and specific ones that had been implemented in existing systems. A topology was proposed based on this review that combined various aspects of these, and improved upon the deficiencies. The proposed architecture design will be detailed in the following chapter.

Chapter 5 The Hardware Design

5.1 Introduction

The state of the art in general floating point DSP processors that are widely available will be analyzed in terms of suitability for the multi-processor system. The processors considered are the TMS320C30 and TMS320C40 from Texas Instruments, and the DSP96002 from Motorola. *Table 5.1* compares the major features for the processors selected. A brief description of each category feature follows. Refer to [Lee87b] and [Lee87b] for an extensive in-depth comparison of these and some other processors available on the market. The nomenclature of [Lee87b] will be used to distinguish between microprocessors and microcomputers; the later have on chip memory whilst the former do not. All of the processors considered here are microcomputers.

5.2 DSP Processor Comparison

Figure 5.1 uses a number of criteria for comparison of DSP processor IC's. It is not always clear upon how to compare some features directly, as different processors may follow different design philosophies. To aid in comparison, a brief description of the categories and their significance to our application will be given.

The MIPS rating (million instructions per second) is the raw processing speed of the microcomputer. The MFLOPs peak rating (million floating point operations per second) is an indicator of the peak mathematical capability. The multiplier category indicates the word size in bits for the integer and floating point hardware multiplier. The ALU word sizes are for integer addition/logical operations and floating point addition operations. The division and square root hardware support capability allows for reduced computation time of division and square roots. Floating point IEEE format number support may be important in some applications interfacing to external data bases or IEEE format devices.

The number of simultaneous internal memory accesses and number of internal data buses give an indication of the internal data bandwidth of the device.

The type of instruction pipeline arbitration is classified as per [Lee87b]. Briefly, interlocked pipelining resolves all pipelining conflicts internally and transparently to the programmer. Time-stationary pipelining gives the programmer more control over the pipelining, commonly by explicitly specifying all simultaneous operations of hardware in one instruction. The former is easier to program while the latter may allow a higher level of optimization, especially for interrupt servicing ([Lee87b], section 2.2)..

Table 5.1: DSP Microcomputer Comparison

Processor	C30	C40	96002
Architecture	Dual External Bus Von Neuman	Dual External Bus Von Neuman	Modified Harvard
MIPS	16.67	25	13.5
MFLOPS Peak	33.3 (e.g. Multiply-Add)	50 (e.g. Multiply-Add)	40.5 (e.g. Multiply-Add-Sub)
H/w Multiply Precision	24 bit Integer 40 bit Floating Point	32 bit Integer 40 bit Floating Point	32 bit Integer 44 bit Floating point
ALU Precision	32? bit Integer/Logical 40 bit Floating point	32 bit Integer/Logical 40 bit Floating Point	32 bit Integer/Logical 44 bit Floating point
H/w Divide & Square Root Support	No	Yes	Yes
IEEE Floating Point Support	No	Single precision IEEE format conversion instructions (single cycle execution)	32 bit single precision 44 bit single extended precision
Registers	8 x 40 bit arithmetic/GP 8 x 32 auxiliary addressing/GP 6 address control 7 other	12 x 40 bit arithmetic/GP 8 x 32 auxiliary addressing/GP 6 address control 7 other	10 x 96 bit arithmetic/GP 8 addressing Register triplets (address, increment, modulo)
Internal Memory	2 x 1k x 32 RAM 64 x 32 Instruction Cache 4k x 32 ROM (factory programmable)	2 x 1k x 32 RAM 128 x 32 Instruction cache 4k x 32 ROM (boot loader)	2 x 512 x 32 Data (X,Y) 1k x 32 Program 64 x 32 ROM (Bootstrap) 2 x 512 x 32 ROM (Sin/Cos lookup)

Table 5.1: DSP Microcomputer Comparison (Continued)

Processor	C30	C40	96002
External Buses	Primary bus: 32 data / 24 address Secondary bus: 32 data / 13 address with separate memory and I/O strobes	i) 32 data / 31 address ii) 32 data / 31 address	i) 32 data / 32 address ii) 32 data / 32 address
Internal Buses	3 data 4 address (plus 4 data in CPU/reg area, 2 data in peripheral area)	same as C30	5 data 3 address
Number of Addressing ALUs	2	2	2
Addressing Arithmetic & Modes	Circular buffer (modulo) Bit reversed Indexed Displacement	Same as C30	Single modulo Multiple register modulo Bit reversed Indexed (extra execution cycle) Linear +/-
Low Overhead Looping Support	Yes	Yes	Yes
Peripherals	2 serial ports 2 timers 1 DMA channel	6 byte-wide inter-processor inter-communication ports (5 Mwords/sec) 2 timers 6 DMA channels	2 DMA channels
External Interrupts	4 General, maskable 1 Reset	4 General, maskable 1 NMI 1 Reset	3 General, maskable 1 Reset
Host Interface and Emulation	Proprietary serial scan interface	JTAG compatible serial scan interface	Proprietary serial scan interface
Parallel System Support	No	Yes	Yes
Bootstrap Loader	No	Yes	Yes

Table 5.1: DSP Microcomputer Comparison (Continued)

Processor	C30	C40	96002
Simultaneous internal data accesses per instruction cycle	3 Data / program fetch 1 DMA	3 Data / program fetch 1 DMA	2 Data 1 Program fetch
Simultaneous internal data accesses per memory block	2 Data/program/DMA	2 Data/program/DMA	1 Data/program 1 DMA
Instruction pipeline arbitration	Interlocked, with some time stationary	Interlocked, with some time stationary	Time stationary

The number of internal registers, the internal memory size, and number of separate internal memory blocks may give an indication of the sustained computational speed of the device. The internal storage is much faster to access than external. The use of physically partitioned internal memory blocks allows simultaneous memory accesses to different blocks due to multiple internal data and address buses.

Addressing modes generally supported are register indirect, immediate indexed and direct. Register indirect can have several variations useful for DSP applications, such as modulo-mode (circular buffer), bit reversed, and arbitrary valued linear increment/decrement. The number of address ALUs to calculate these values can affect the sustainable calculation speed. There are various restrictions not mentioned here in all DSPs that can affect the utility of these special addressing modes. Another addressing mode of great advantage for DSP work is low overhead looping, where a single or block of instructions can be executed a pre-specified number of times without incurring the overhead normally associated with branching (e.g. instruction pipeline flushing).

The number, size and speed of the external buses can affect the performance and system design efficiency. The number and type of internal peripherals can reduce external system complexity, and increase system performance.

The host interface type, bootstrap loader support, and the associated parallel system development support can affect system usability in terms of additional user written software required and debug effort to utilize the system fully.

5.3 Processor Choice

The TMS320C30 was chosen as the processor to be used in the design based on the criteria of availability, performance, development support tools, and the host/emulator interface.

5.3.1 Why Choose the C30

Of the processors considered above, the C30 was the only one available at the time of the hardware design (fall of 1989). It was just about to become available in sample quantities, and full device specifications had been published. The Motorola 96000 was six to nine months away from sampling (barring any delays) and no published detailed specifications were yet available. Obviously, the TMS320C40 was in preliminary design as it did not sample until the fall of 1991.

The performance of the C30 is similar to 96000 in terms of raw processor speed, bus structure, features, and internal architecture. The C30 may have more features and be easier to program than the 96002, but the latter may be faster because the code could be optimized due to the greater control available over the instruction pipeline and parallel operations.

The development tools and software support appear to be more developed than those of Motorola. There is an integrated development tool package from TI, consisting of ANSI C compiler, assembler, debugger/emulator, simulator. Another factor is the previous experience Carleton University has had with TI's TMS320C10/C25 hardware and software development tools.

In terms of the host/emulator interface, all processors were similar. The C30 has a proprietary serial scan interface. This results in a simple connection to PC host via emulator, and this emulator can be easily expanded to access multiple C30s (sequentially) via an electrical buffer arrangement. This allows program loading and control of a multiple C30 environment. The 96000 also has a proprietary serial scan interface. The C40 has serial scan interface, but is fully JTAG compatible to allow multiple devices to be connected in serial, i.e. no extra hardware to be added for parallel systems. The emulator software is designed to support the control of multiple C40s simultaneously.

One important feature of the C40 that is being touted for multi-processor systems is the intercommunication ports. Each C40 is endowed with six byte wide communication ports, enabling direct inter-connection with six other C40s in a multiprocessor system. The ports operate at a peak rate of 20 MBytes/sec (equivalently 5 MWords/sec; 16⁶)

MBits/sec). The data is written and read from memory mapped registers. There are intermediate FIFOs and interlock control hardware to prevent data loss and problems associated with asynchronous operation. There are also six independent DMA channels to allow transfer of port data to and from memory without CPU intervention. As such, the data can appear to be written from one C40 into a connecting C40's memory space, as long as the data communication consists of predefined block sizes and ordering. The port data does not implicitly contain a destination address, and writing to random ordered addresses in the target C40's memory space would require explicit intervention by the sending and target C40s.

The communication ports are useful for systems of seven or less C40s with rigidly defined data transfer requirements. If a larger number of processors are required, generally CPU intervention will be required in processing the intercommunication data traffic, unless additional hardware is utilized. An example would be a ring bus using one or more (up to three) ports to connect to adjacent C40s. To send data, the CPU would have to construct a data 'packet' consisting of a destination address, one or more data words, and any control parameters, and perform multiple writes to the appropriate port(s). Incoming data would be analyzed by the CPU to see if it was for itself, and either write the data into its own memory or write it to the other ring bus port(s), accordingly. This processing obviously decreases the available CPU processing capability, and this effect worsens as the data inter-communication traffic is increased even if the data is not address to the intermediate C40. For larger multi-processor systems, the C40 communication ports would not seem to alleviate the data intercommunication problems without additional hardware solutions.

With the choice of the C30 and the previously discussed slotted- ring bus architecture, a system block diagram can be constructed. *Figure 5.1* illustrates the principal components in the system. The C30s are connected to the ring bus through an Input/Output processor (IOP) that handles intercommunication over the unidirectional ring bus. The host PC connects to the C30s via an emulation bus and the associated emulation bus controller. The emulation interface board is the Texas Instruments XDS-500, a commercial product that is bundled with TI's emulator/debugger and C/assembly software..

5.3.2 C30 Detailed Description

The TMS320C30 DSP processor that forms the core of the PE has many features that are suitable to this application. Many of the key features have previously been listed in *Table 5.1*. A brief summary of C30 features is reiterated here:

Figure 5.1 C30 Multiprocessor Processing Element (PE) Block Diagram

- dual 32 bit external data buses.
- 60 nS instruction cycle time.
- floating point ALU.
- most instructions are single cycle, with two parallel operations in one cycle possible (e.g. simultaneous multiply and add).
- internal 2k RAM, ROM, and instruction cache.
- internal peripherals (DMA, timers, serial ports).
- auxiliary register arithmetic units for parallel addressing calculations.
- serial scan emulation interface.

Some of the unique features of the C30 that are important to our application:

- parallel instructions targeted towards DSP algorithms, e.g. Multiply/Accumulate, Arithmetic/Store, Logical/Store, Load/Load
- addressing modes and hardware for DSP work:

- address ALUs for concurrent address arithmetic
- circular buffer mode (modulo address arithmetic)
- bit reversed mode (for FFTs)
- single instruction and block repeat (low overhead looping)
- high internal bus b/w
- instruction pipeline (4 level) with transparent pipeline conflict resolution
- interlocked instructions
- interrupts

The C30 User's Guide [*Texa90b*] provides an extensive description of the C30 microcomputer features.

5.4 Slotted Ring Bus Features and Protocol

The data intercommunication bus is a logically separate entity from the CPU portion of the PE. The I/O processor (IOP) controls the operation of the bus and operates fully asynchronously to, and independently of, the CPU (except for bus configuration, and global bus control e.g. bus reset). The bus and processor are temporally decoupled using First-In-First-Out (FIFO) buffers within the IOP as shown in *Figure 4.3* and *Figure 5.2*, minimizing their interaction. This concept is similar to a number of traditional hardware design concepts:

- the separate computational and bus interface units inside many microprocessor ICs (e.g. Intel 8086).
- the use of a cache memory and it's controller to decouple the interaction of a fast CPU and slow memory.
- smart I/O peripherals (e.g. DMA controllers, dedicated I/O processors such as the Intel 8089, caching hard disk controllers, etc.).

To the C30 CPU on the PE, transferring data to and from another PE appears as writing to and reading from zero wait state memory. The actual transfer of data is transparent to the CPU i.e. it doesn't have to wait while the data transfer physically occurs. The intercommunication process itself can be subdivided into two functions:

- the actual transfer of the data between PEs.
- the program synchronization mechanisms offered by hardware support (external flag

Figure 5.2 Three PE Ring, with IOP Structure Shown

and signalling bus). These are discussed in the following sections.

The inter-processor communication and program synchronization methods are unique to the multi-processor system presented. As illustrated in *Figure 5.1* and *Figure 5.2*, the processing elements are interconnected in a ring type architecture. The IOP bus interface operates independently and synchronously with respect to the CPU. In essence it is a direct memory access (DMA) processor type of interface that automatically handles the transfer between PEs when a request for data transfer is issued by the CPU. The ring type architecture exhibits the following attributes:

- processing modules that communicate in a nearest neighbor sense have a low communication latency time while those increasingly further in distance have a correspondingly longer latency.
 - This tends to match the principle of communications locality observed in algorithm partitioning as previously mentioned.
- the low physical bus loading allows fast operating speeds, and is technology dependent allowing future upgrades in speed. The fast speed serves to minimize the inherent latency of the ring bus.

5.5 Slotted Ring Bus Hardware Design

A more detailed description of the proposed bus design will be stated. This will cover both the operation and physical modules that the bus is comprised of.

5.5.1 Bus Data Transfer Timing

Each PE writes to the bus in an asynchronous manner with respect to the bus operation, thus the PE does not wait for the transfer to complete. An output FIFO is used to interface the processor to the bus to absorb any timing differences (*Figure 5.2*). The C30 writes to an address in its memory address space into the bus output FIFO. The data and address wait at the output of the FIFO for an available bus slot to transfer the data to the target PE.

The bus transfer is logically divided into discrete time slots, resembling a multi-bit data shift register. Each stage of this shift register is synchronously clocked by a common bus clock. The time slots are further grouped into frames, delineated by a frame sync bus signal. The time slots can be configured to be either output slots or input/pass slots. Generally, the system is setup so that there are N time slots in a N PE system. The first time slot is assigned as an output slot and the rest as input/pass slots. During output slots, any incoming bus data (i.e. from the bus input register stage) is overwritten with the data and address appearing at the output of the bus output FIFO. This is done by selecting the bus output FIFO via the bus output multiplexer. Thus all PEs physically write their data at the same time onto the bus, synchronized to the bus FRAME SYNC signal.

The input/pass slots simply pass data along the bus from the PE bus input to the bus output, via the input register i.e. the PE's bus output is selected to be the data clocked in from the bus input from the previous bus time slot using the bus output multiplexer. The net effect is that the data is shifted down the bus by the bus clock. During each input/pass slot the bus control circuitry monitors the incoming data to see if it is addressed for itself (either it's unique bus address or a bus broadcast address). If it detects such a condition, the bus interface will 'pick off' the data present on the bus at that time slot. This 'picked off' data is passed through an input FIFO and is then synchronously (with respect to the communication bus) written into one side of a dual port memory. The lower part of the address information accompanying the input data is used to select a unique address in the DPRAM to be written to. The C30 CPU of the PE is able to access the other side of the dual port memory and thus read the data that was sent to it.

The bus transfer is unidirectional both logically and physically. Each PE can only write data directly to another PE; it cannot directly read from another PE's memory. Thus the data transfer is logically unidirectional. The physical transfer of data along the bus occurs in a predetermined direction, as determined by the physical interconnection of the PEs.

Since the interconnected PE bus appears as a shift register, there is a latency asso-

ciated with the transfer of data between PEs. The nearest neighbor in the direction of physical bus data transfer will have a short latency, while the nearest physical neighbor in the opposite direction will have the longest latency. Since the bus can be made quite fast (currently 30 nS per time slot) due to the short interconnection lengths, and the system is designed for a small number (less than 16) of PEs, the maximum latency can be kept small and compares favorably with that of a global arbitrated bus structure. For comparison the VME bus has an absolute data transfer period of 115 nS and a minimum bus arbitration period of 90 nS. It should be noted that these values are the absolute minimum as stated by the VME specification, and are rarely achieved in practical systems.

5.5.2 IOP Bus Interface Controller

As the name implies, the bus controller generates signals to control data PE and the bus. It can be logically divided into three parts as shown in *Figure 5.3*: bus timing generation, bus input/output processing, and DPRAM control..

Figure 5.3 Bus Controller Components

The two major bus control signals are the bus clock (BCLK) and bus frame sync (BFS). These are distributed to all PEs in the system by the bus clock generator module within the MPU system controller module (see section 5.5.3 "*Multiprocessor System Backplane and Controller Module*").

The bus timing logic counts the time slot number, and creates control signals for

the other bus interface controller logic blocks. The control signals are based on the time slot number and the user programmable information contained in the time slot control RAM. Its contents define which time slot numbers the IOP is allowed to read data from the bus into the bus input FIFO, pass data from the bus input register to the bus output, and output data onto the bus from the bus output FIFO. There are also four user programmable tag bits that are used by the input/output processing logic as an address matching value.

The bus input/output processing block consists of three subfunctions. The bus control subfunction matches the incoming bus address to the address tag bits, and provides enabling control signals to the bus input FIFO state machine if appropriate. It also provides timing and enabling signals to control the operation of the bus input and output state machines, and harmonized control signals to the bus output multiplexer. The bus input and output subfunctions are state machines that generate a sequence of FIFO control signals based on the FIFO status and the control signals from the bus control subfunction. The bus input state machine generates signals to transfer incoming bus data from the input register output to the input stage of the bus input FIFO. The bus output state machine produces signals to transfer bus output data from the bus output FIFO output stage, through the bus output multiplexer, to the PE bus output.

The DPRAM controller monitors the status of both the bus input FIFO output stage and the left port of the DPRAM. Based on these and the timing control signals, it generates a sequence of FIFO and DPRAM control signals to transfer the incoming bus data from the bus input FIFO's output to the correct address within the DPRAM.

5.5.3 Multiprocessor System Backplane and Controller Module

The backplane was designed in a physically interleaved PE placement. This means that logically adjacent PEs on the ring are actually placed two backplane slots apart. The inter-PE data flow skips every second position in one physical direction, turns around at the end of the backplane, and then flows back in the other physical direction connecting alternate previously unused positions. Thus the physical numbering of logically numbered PEs on a six PE ring would be 1-6-2-5-3-4. This is done so that there is no long return path required to join the "ends" of a linearly drawn ring bus, which would adversely affect bus performance.

The components of the system control module are illustrated in *Figure 5.4*. It consists of the bus clock generator and the C30 emulation bus control logic. It physically resides in the card cage next to the backplane and the power supply.

The bus clock generator produces eight bus clock signals and eight bus frame sync

signals. The bus clock is a square wave of period 30 nS (33.3 Mhz), and the frame sync is 30 nS pulse that frames a bus clock rising edge. The frame sync is repeated every N-1 bus clock cycles, where N is the number of processors in the system. They are distributed by a parallel low skew driver/cable distribution system, to ensure all PEs receive the clock and frame sync edges within a known, tight tolerance of each other

The emulation bus controller logic is simply a demultiplexer that decodes four bits of the PC host's parallel port into a one-of-sixteen PE selection signal for the emulation bus electrical buffer. This allows the PC to select one processor in the system to electrically connect to.

Figure 5.4 MPU System Control Module Block Diagram

5.5.4 Inter-processor Synchronization Hardware

Barrier synchronization between processors is supported by three hardware mechanisms:

- bus interrupts.
- external flag bus (BUS_XF*).
- dual port RAM signalling.

Each will be discussed in detail below.

5.5.4.1 Bus Interrupts for Barrier Synchronization

The interrupt bus can be used to perform barrier synchronization in a number of ways. The simplest is to generate periodic interrupts via an external interrupt source driving a bus interrupt lines. It simultaneously interrupt all the processors in the system. This is similar to the normal "baud interval" signal processing found in many DSP applications of interest. It is assumed that the time between interrupts is sufficient for each processor to finish all calculations, and for all data transfers between processors to be completed. If not, then catastrophic failure of the program run is undoubtedly the result. This is not a very user friendly method of implementing barrier synchronization.

Figure 5.5 Bus Interrupt Block Diagram

Another method would be to have the most heavily loaded PE in the system generate the interrupt. Since it is the most heavily loaded, by the time it is finished its calculations and intercommunication, all of the other PEs will have finished their computations and intercommunication. However, it is up to the user to determine the most loaded PE in the system, and if done incorrectly will again result in catastrophic program failure. Neither of the former methods are recommended.

A variation of the latter method is to arbitrarily assign one PE in the system as the 'master'. It starts the system computing, and waits for all of the other PEs in the system to indicate to it (via some protocol set up by the programmer) that they are finished computing. When all PEs have reported they are finished, the master PE starts the next computational cycle by generating a bus interrupt. This method has the significant advantage of

being self-timed. It will only perform the next computational cycle when all of the PEs are finished, no matter how long they take, or if the times vary from cycle to cycle. The disadvantage is the programming and execution overhead of the extra intercommunications required to implement this method, as well as the latencies introduced by servicing interrupt service routines (ISRs).

5.5.4.2 External Flag Barrier Synchronization

This bus is a one bit bus that runs in parallel to all of the PEs in the system (see *Figure 5.6*). This bus is unique in that it employs hardware/software synchronization capability built into the C30. The C30 has two pins dedicated to allow hardware control of the program using special "interlocked" instructions (pins XF0, XF1, see section 7.3 of the C30 users guide [Texa90b]). The PE hardware has combined the two flags into one open collector signal (BUS_XF*) that is connected in parallel between all of the PEs. Each PE can hold the BUS_XF* line active (i.e. physically low voltage) via an open collector driver by setting the C30's XF0 pin, and can sense the state of BUS_XF* via the C30's XF1 pin state. The interlocked instructions of the C30 (LDFI, LDII, SIGI, STFI, STII) can be used to halt processor execution until the BUS_XF line is inactive. This special halt ability is important because it does not flush the C30 instruction pipeline, thereby allowing the C30 to respond to the state of BUS_XF* immediately (i.e. within one bus cycle due to the asynchronous nature of the system). This is in contrast with the much longer interrupt response time due to the interrupt function call and return overheads.

Figure 5.6 External Flag Synchronization Bus

The sequence of events followed by each processor to utilize the BUS_XF* feature

is:

- i) initialize the BUS_XF* line to the hold state
- ii) perform all of the necessary processing for the assigned task, including send any data to the destinations required (with acknowledge if required)
- iii) set the local BUS_XF* control to the ready state
- iv) by executing one of the special interlocked instruction, monitor the state of the BUS_XF* flag and don't proceed until the it becomes inactive
- v) when the BUS_XF* state becomes inactive, it means that all of PEs have set their local BUS_XF controls to inactive, indicating all PEs in the system are done computing their task
- vi) set the BUS_XF* to the hold state and proceed to the next assigned task (in many cases this is to simply repeat the same task)

In practice, this is implemented in two or three assembler instructions (a single function call in C). This sequence of programming steps forces all of the PEs to wait for the slowest PE in the system to complete its task before any of them can proceed to the next task. In this manner the BUS_XF hardware feature implements barrier synchronization.

5.5.4.3 Dual Port RAM (DPRAM) Signalling Synchronization

Associated with each port of the dual port memory is a unique pair of address locations that control the state of the signalling flags. The reading and writing to these locations control the state of two hardware lines, as outlined below. One hardware flag is associated with each port of the DPRAM, and thus each of their associated control locations are similarly associated with a DPRAM port. The control addresses will be designated the signalling flag control address locations, and the hardware lines the signalling flags. This function is provided by the specific DPRAM IC's used in the hardware design of the PE. The DPRAM configuration and the associated signalling flags are illustrated in *Figure 5.7*.

When a control location is accessed through the memory port on one side, the signalling flag of the opposite port is set. When the reset location is accessed, the signalling flag on the same port of the memory is reset. Since the bus side of the memory can only be written to by the bus, the processor side signalling flag can be set by another PE writing to the appropriate bus side memory port. The PE can reset this flag at will by accessing the reset port on it's memory port i.e. the processor can set the bus side port signalling flag by accessing the set location on it's memory port. Since the other PEs in the system cannot directly determine the state of this flag, a separate signalling bus was implemented to allow each PE to read the state of every other PE's bus side signalling flag

Figure 5.7 Signalling Synchronization Bus Structure

5.5.4.4 The Signalling Bus

This bus is very similar in structure to the main slotted ring bus but is only one data bit wide, with associated clock and frame synchronization lines. It is a ring structure that is temporally partitioned into discrete time slots by using a shift register bus structure. Each PE's signalling bit is assigned to a time slot. A frame synchronization signal occurs every N time slots, where N is the number of PEs in the system. All PE's output their signalling bit onto the bus during the frame synchronization time slot. This signalling data is then shifted to each succeeding PE by the bus clock through the bus register structure. As the data from each PE passes through the signalling bus interface of each PE, the data is shifted into an internal shift register. The signalling bus interface structure of a single PE is shown in Figure 5.8. When the shift register has clocked all of the signalling bits in from every other PE in the system (as indicated by the next occurrence of a frame sync.), the contents of the shift register are transferred to a parallel register which can be interrogated at will by the CPU. The signalling data in the parallel register can also be used to interrupt the CPU in a mask programmable fashion. The PE's own bus side signalling bit value is included in this signalling status word, allowing read-back of its own bus side DPRAM signalling state.

The result is that a PE can set the C30 side signalling flag on another PE, and that PE can reset it. A PE can also set the bus side signalling flag of its own DPRAM, and its

Figure 5.8 Detailed Signalling Bus Interface Block Diagram

value is automatically distributed to all other PEs in the system, and any of the other PEs can reset the state of that specific flag. A PE may also allow the bus side signalling flag state of any other PE to cause a local interrupt.

5.5.5 Bus Performance

The bus is designed to operate with a minimum clock period of 30 nS (33.3 MHz maximum). During this time period in the first time slot after the bus frame sync, each slot can be filled with a data packet from a different PE, and then shifted along the bus on subsequent periods of the bus clock. The bus is reloaded with new data upon the next frame sync and the shifting repeated. Each data packet contains 32 bits of data (four bytes) and 15 bits of address. Due to the shifting, a bus communication latency is introduced. The actual bus bandwidth is dependant upon how the bus is configured in terms of it's frame sync length. In the general case (same as the worst case) it frame sync and bus are set up to allow communication between any PES. A best case peak data transfer rate can be obtained when the data flow is unidirectional to the adjacent PE only. Then the bus can be

PE every bus slot (but not further than the adjacent PE).

In the general case, the bus frame length is the same as the number of PEs in the system i.e. there are N PEs and M bus time slots, where $N=M$. Each PE may write a data/address packet to the bus during the first time slot of a frame and then it is shifted $M-1$ times on the bus so it passes by each PE. During the first time slot, N packets are written, and then they are shifted for $N-1$ slots for a total of $M=N$ slots. Thus data is transferred at a rate of $N/M = N/N = 1$ packets per time slot. The data transfer rate can be stated in a variety of ways, depending upon what type of system it is being compared to. For comparison to parallel data buses, the transfer rate is usually stated in data words or bytes per second. The transfer rate of this bus would be:

$$1 \text{ word}/30 \text{ nS} * 4 \text{ bytes/word} = 33.3 \text{ Mwords/S} = 133.3 \text{ Mbytes}$$

When considering serial buses, the rate is usually stated in bits per second. This is the raw bit speed of the bus. For fair comparison between parallel and serial buses, the address bits should be included in the calculation when converting from a parallel bus speed to an equivalent serial bus speed. Accordingly, the transfer rate of the bus is

$$(32 + 13) \text{ bits}/30\text{nS} = 45 * 33.3 \text{ Mpackets/S} = 1.4985 \text{ Gigabits/S}$$

The inclusion of the address bits in this calculation is fair because serial links must include address information pertaining to the destination of the packet of information.

Table 5.2 shows the latency of communication between PEs for the minimum, average, and maximum cases. This table states the elapsed time from when a source processor writes a data item to it's bus output FIFO, until the time that data is received in the local DPRAM of the destination processor D logical positions away on the ring bus, in a N PE system..

Table 5.2: Latency for Furthest Intercommunication Path in a N PE System

Delay for:	Minimum	Average	Maximum
Waiting for output slot	0	$N/2$	N
Time through bus output FIFO	1	1	1
Transport via bus	D	D	D
Time through bus input FIFO, write to DPRAM	2	2	2
Total (Bus cycles)	$D + 3$	$N/2 + D + 3$	$N + D + 3$

In *Table 5.2*, D is the distance along the bus of the destination PE (in the direction of the bus shifting) and N is the number of PEs installed in the system. The minimum distance is to the next PE in the direction of the bus data flow. The maximum distance is to the adjacent PE in the direction opposite the bus data flow, and the average is for communication to the PE $N/2$ positions away. *Table 5.3* gives the average latency in bus cycles and time for various numbers of PEs installed in the system (N) and communication destination distances (D). Note that the table gives the bus latency only, and does not include the time for the C30 to read and write to memory locations (which is the same as any other zero wait state memory access).

Table 5.3: Average Intercommunication Latency for Varying Number of PEs in a System

Number of PEs N	2 PEs (bus cycles / nS)	4 PEs (bus cycles / nS)	8 PEs (bus cycles / nS)	16 PEs (bus cycles / nS)
Intercommunication Distance D				
2	6 / 180	7 / 210	9 / 270	13 / 390
4	n/a	9 / 270	11 / 330	15 / 450
8	n/a	n/a	15 / 450	19 / 570
16	n/a	n/a	n/a	27 / 810

The best case arises when all data flow is unidirectional between adjacent PEs. Although this is highly application dependant, it is quite common in DSP algorithms i.e. pipelined data flow. In this case the bus can be configured for one time slot per frame, ($M=1$) with data being output onto the bus from each PEs bus output FIFO during the slot, and clocked into the successive PEs input register at the end of the time slot. This N PEs can write N data packets per frame, and the number of packets per frame is then $N/M = N$. Using the same nomenclature as for the general case, the peak data transfer rate can be stated as:

$$N * 33.3 \text{ Mwords/sec, or equivalently as}$$

$$N * 133.3 \text{ MBytes/sec, or}$$

$$N * 1.5 \text{ GigaBytes/sec}$$

Table 5.4 shows peak transfer rates for various numbers of PEs (N) in a system.

The minimum latency for a data transfer between adjacent PEs is calculated as:

Table 5.4: Peak Data Transfer rate for Various Number of PEs in a System

Number of PEs (N)	Word Transfer Rate (MegaWords/Sec)	Byte Transfer Rate (MegaBytes/Sec)	Equivalent Serial Transfer Rate (Gigabits/Sec)
2	66.7	267	12
4	133	533	24
8	267	1,067	48
16	533	2,133	24

Table 5.5: Adjacent PE Data Transfer Latency

0	wait for bus output cycle
1	through the bus output FIFO
1	bus transfer time
2	bus input
4	bus cycles (latency = 120 nS)

5.6 Processing Element (PE) Features

Each individual PE is a self contained processor consisting of (*Figure 5.9*):

- the TMS320C30 DSP oriented central processing unit (CPU)
- a local memory section consisting of 128K by 32 of random access memory (RAM) and 8k by 32 of erasable programmable read only memory (EPROM)
- control logic for external bus and signalling interrupts
- trace buffer
- parallel CPU host interface
- an inter-processor communication bus interface, consisting of
 - data transfer mechanism, including globally mapped dual port RAM (DPRAM) memory
 - program synchronization mechanisms (signalling bus, external flag)

Each will be discussed in turn below.

Figure 5.9 Processing Element Block Diagram

5.6.1 Central Processing Unit (CPU)

The TMS320C30 CPU features were previously discussed in section 5.3.2 "*C30 Detailed Description*" and are further detailed in reference [Texa90b].

5.6.2 Local Memory

The local memory consists of up to 128K by 32 zero wait state static RAM and up to 8K by 32 bit zero wait state EPROM. An expansion connector is provided to allow piggy-backed circuit cards to be easily added for additional memory and other peripherals.

5.6.3 Trace Buffer

The trace buffer can be used for temporary storage of interim data for purposes of assisting program debugging or temporary FIFO data storage. The buffer is 32 data bits wide plus four address tag bits. It is self emptying i.e. it always accepts new data, and discards the oldest data if it is full after a write to it.

In this way the last N data items are kept, where N is the physical FIFO IC depth (varies between 512 and 4k, depending on the actual chips installed). Also written into the trace buffer are the lowest four address bits, which act as a tag to identify the data. This allows up to sixteen unique data streams to be stored and then identified at a later time.

5.6.4 Interrupt Control

The interrupt control circuitry provides for a number of possible interrupt sources

- four bus interrupts.
- the PE side interrupt of the local dual port RAM (DPRAM).
- the bus side DPRAM interrupts from the other PEs in the system (their status arrives via the signalling bus and are fully maskable).
- the expansion bus interface.

All of the interrupts are individually maskable, and the four bus interrupts can be activated from each individual PE. This allows a PE to interrupt the system for control, error flagging, and other purposes. An important consideration in interrupt usage is the execution overhead due to the interrupt vector fetch, and the interrupt function call and return. The C30 instruction pipeline is flushed twice in this sequence, pipeline delays are added, and a total overhead of ten to eleven instruction cycles occurs.

The C30 interrupt input pins are level sensitive. This requires that pulse be applied to the input pin if one and only one interrupt is to be generated. To achieve this, an asynchronous (to the local PE clock) bus interrupt will be processed by the LCA circuits to produce a 1.5 CPU cycle length interrupt on the CPU interrupt input, synchronized to the CPU clock. The latency introduced by the LCA circuitry is .5 CPU cycles to allow for C30 input setup. An additional interrupt processing latency of zero to one CPU cycle is introduced, due the asynchronous nature of the bus interrupt with respect to the CPU clock. The interrupt latency of the CPU¹ is four CPU cycles before the interrupt vector is read, six cycles before the first instruction of the interrupt subroutine (ISR) is read, resulting in nine cycles before it is executed. This is due to the requirement of flushing and then refilling the internal instruction pipeline, and these may be modified by the internal cache operation. According to the same source, the interrupt at the CPU input pin is timed from the falling edge of the H1 clock phase (when the setup time is met), thus the interrupt pulse produced by the LCA should start on the rising edge of H1 for a minimum latency of one half of a cycle. The total latency of the system to a bus interrupt is calculated as

1. figure A-15 page A-19 of the C30 Users Guide [Texa90b]

Table 5.6: C30 Interrupt Processing Latency

1.0	due to asynchronous clocks
1.0	LCA processing latency
.5	CPU interrupt input setup
9.0	CPU processing latency
11.5	total system interrupt processing latency (in C30 instruction cycles; ==690 nS)

5.6.5 Parallel Input/Output Control Register

The parallel input/output control register is written to and read from by the C30. The outputs are used to control various on-board functions such as memory bank switching, and communications bus mode and control. The inputs are used to monitor the trace buffer and other on-board hardware conditions. There are also spare (uncommitted) inputs and outputs available to the user.

5.6.6 Parallel C30 Host Interface

The host interface utilizes the Texas Instruments C30 emulation circuit card and accompanying software. The card itself allows full in-circuit emulation of a C30 by connecting to the C30 IC and controlling it via a serial scan path. In effect the C30 itself becomes the emulation pod. Functions supported are down-loading data/programs into any memory location of the C30 address space, examination of all C30 registers and memory (both internal and external), and controlled execution (e.g. single step) of a C30 program. This emulation system was originally intended for single processor applications. To allow it to interface with the multi-processor system, each PE is provided with three-state buffers that allow the emulation system to be logically connected to a single PE at a time, the PE selection being under control of the host PC as shown in *Figure 5.10*. By allowing the host to enable one PE emulation interface at a time, the emulator is able to control multiple PEs in a sequential manner.

5.6.7 Interprocessor Communication Bus

The intercommunication bus structure and operation were previously discussed in *Section 5.4* and *Section 5.4*.

Figure 5.10 Parallel C30 Emulator Interface Bus

5.7 Conclusion

Current DSP processor ICs were compared and a suitable one chosen. A detailed description of the slotted bus implementation and expected performance was provided. The program synchronization hardware was also described, as were additional hardware features of the PE design.

PAGINATION ERROR.

ERREUR DE PAGINATION.

TEXT COMPLETE.

LE TEXTE EST COMPLET.

NATIONAL LIBRARY OF CANADA.

BIBLIOTHEQUE NATIONALE DU CANADA.

CANADIAN THESES SERVICE.

SERVICE DES THESES CANADIENNES.

Chapter 6 Hardware Verification and Performance Evaluation

6.1 Introduction

The actual multiprocessor system was constructed. It consists of four identical PEs, and includes a backplane design that can accommodate up to 16 PEs. The backplane resides in an 19 inch rack mountable card cage (9U physical height). The card cage also accommodates the power supply and the system controller module. Next, low level software tools were created to allow testing and programming of the system.

The hardware verification consisted of two phases. First the hardware was functionally tested to ensure that it did what was expected in the logical sense. Next the design was subjected to performance testing to ensure it performed within the design timing specifications. The algorithms used for the benchmarks were the HDSL and Mel-Cepstrum examples previously introduced in chapter 3.

Furthermore, a second complete system was constructed as part of an independent contract awarded to Carleton University.

6.1.1 Functional Verification

The functional verification and testing proceeded in stages. A single PE was built up section by section, and each major on board function tested, with exception of the IOP. A second PE was then built, and the IOP bus interface circuitry was tested. A modular expandable 8 slot backplane printed circuit board (PCB) was designed and built, along with two additional PEs. Using the resulting four PE system, the IOP was tested more extensively. The program synchronization hardware features and the signalling bus were also tested at this time.

6.1.1.1 PE Testing

The host interface was the first part of the system tested, as dictated by virtue of it's being the C30 chip emulator. The PE host controller was tested to select one PE at a time, and both the test and bus emulator ports were ensured to operate properly with a fully loaded backplane.

Testing of the primary memory was accomplished by standard bit pattern testing of the addressing functions (e.g. walking ones, etc.) and data words. The DPRAM was tested from the C30 side in the same manner as the primary memory. The bus side port of the DPRAM was tested as part of the intercommunication bus.

Peripherals testing includes the trace buffer, the interrupt control circuits, and the parallel I/O functions. The trace buffer had extended testing of fill/self empty feature, including read-back and verifying the operation of the address tags. The external bus interrupt and signalling bus interrupt pulse generator circuits were checked to ensure a 1.5 bus cycle pulse is generated to the C30, as required by its specifications. This length of pulse is required due to the C30 using level sensitive interrupts (see [T190] section 6.6). The bus interrupts were extensively tested as part of the interrupt based barrier synchronization testing. The parallel I/O outputs were verified by toggling all output control bits and the input checked to ensure all input bit states can be read in either state.

6.1.1.2 IOP Bus Interface Testing

The bus control sequencer was verified as being properly programmable and the signal output sequences operating. The bus was then wrapped around to itself (i.e. BUS_OUT connected to BUS_IN on an single PE) and data communication was tested extensively.

A system was then assembled with two PE's and a wire-wrap backplane. The intercommunication of data blocks and the bus addressing were tested extensively. The bus period was shortened slightly to 26 nS (from a design nominal of 30 nS) to ensure there was sufficient margin for timing variations. It operated at this speed for over 2^{32} transfers of blocks of size 1022.

A four PE system on a printed circuit backplane was assembled and tested. It operated at design speed for 2^{32} transfers of blocks of size 1022. The data was transferred from PE to PE in a pipelined fashion with the data being inverted at each stage. Each block had a unique data starting value, with successive values in the block incremented by one. Block transfers with other data patterns such as 00000000h, FFFFFFFFh, AAAAAAAAh,

5555555h were tested for extended periods to test for situations where errors might be caused by a large number of bits changing states simultaneously.

6.1.1.3 Signalling Bus

The signalling bus was tested by accessing the DPRAM control locations to set, reset the bus side signalling flag. The state of the bits were verified to be reflected (i.e. properly distributed to) other PE's in the system. The interrupt generation, masking, and pulse generator were verified.

6.1.2 Hardware Performance Verification

The performance of the bus interface was of primary concern in terms of the data transfer speed. The performance of the PE functional blocks was tested previously while performing the functional tests at the C30 processor full clock speed. The basic bus timing was also verified during the IOP functional testing. The bus data transfer latency was verified to be within the expected number of bus clock cycles, as stated in the system design specifications (section 5.5.5 *Bus Performance*, on page 69).

6.2 Algorithm Performance Evaluation

Two different DSP system algorithms were tested on two different system configurations:

- a MEL-Cepstrum speech processing front-end on a two PE system, relative to a 33 MHz 386/387 PC work-station.
- a HDSL transmitter/channel simulator/DFE based receiver on a four PE system, relative to a SUN SPARC I work-station.

The execution times and relative speedup values are reported for the work-station, single C30, and multiprocessor executions. Results are also reported separately if optimized functions (i.e. assembly language code) were used for some of the functions. Intercommunication overhead was noted when significant.

The general method of porting these applications from the work-station environment to the multiprocessor is illustrated in *Figure 6.1*. The algorithms were first debugged on the work-station environment and then modified to run on a single PE. In the case of the HDSL program, C compiler macros were used in such a way that the same program

was used in both cases, simply by recompiling for the target machine. Next the algorithm was partitioned onto the multiprocessor by splitting it into approximately equal execution time segments (based on an execution profile performed on the work-station), and the appropriate data intercommunication and program synchronization functions were added to the code. The code is tested for functional correctness, and then the performance is noted. If inadequate, load balancing and optimized assembly language functions are used to improve the performance to an acceptable level.

Figure 6.1 Software Development Cycle

6.2.1 Program Synchronization

The synchronization of data flow chosen for these examples was a simple scheme using both the barrier synchronization and messaging interrupts. The general program execution is illustrated in *Figure 6.2* for data intercommunication between two PEs. No attempt was made to overlap the communications with computation using the C30 DMA controller (this technique is illustrated using this system by Oakley [Oak191a] in his application). The whole system performs no useful computation during the data transfer interval, which leads to a lower effective R/C ratio. As mentioned, this inefficiency can be virtually eliminated by overlapped computation/communication techniques such as double buffering and DMA initiated transfers.

Figure 6.2 Program Synchronization of Data Flow

6.2.2 MEL-Cepstrum Speech Processing Front End

The MEL-Cepstrum algorithm is a block oriented processing step used in speech recognition systems. The algorithm particulars are given in references [Davi80] [OSha90a] [Pars87a]. The two PE partitioned block diagram is shown in Figure 6.3. The speech samples are assembled into blocks of 256 samples, and blocks are processed with a 50% overlap. A significant portion of the processing time is consumed in the vector dot products and the FFT calculations. Since they are standard DSP building blocks, they are likely candidates for reusable, optimized, assembly language coded functions.

Figure 6.3 MEL Cepstrum Block Diagram Partitioned for Two PEs

Table 6.1 shows the execution times for the MEL cepstrum algorithm. The base line workstation was a 33 MHz 386/387 based PC. The C code run on multiprocessor was

2 OF/DE 2

PM-1 3 1/2" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

not optimized in any way, although it would be possible to use standard optimized functions for the FFT and cepstrum calculations. The optimization was not performed because real time operation was reached with only two PEs running the straight C code port. In the two PE case, there is a 0.8 mS block transfer time as part of the 7.7 mS execution time listed. By using double buffering and DMA techniques for the intercommunication, it would occur in the background and not extend the execution time as in this example

Table 6.1: Execution Time Comparison for the Mel-Cepstrum Algorithm

System	Execution Time (mS/Block)	Times Real-Time (R.T.=8 mS/Block)	Improvement Ratio ^a	Intercommunication Time (mS/block)	R/C Ratio
33 MHz 386/387	79.3	9.9	1.0	n.a.	n.a.
Single PE	13.75	1.7	5.75 (1.0)	n.a.	n.a.
Dual PE	7.7	.96	10.3 (1.79)	.8	9.625

a. The first value is w.r.t. the workstation, the second value w.r.t. the single PE case

6.2.3 HDLS Transmitter-Channel-DFE Receiver Simulations

The next benchmark examples deal with the HDLS algorithm. The first is a simple HDLS transmitter/DFE receiver implementation without any crosstalk or adaptive filters as shown in *Figure 6.4*.

Its execution times are shown in *Table 6.2*, with respect to Sun 3/60 and SPARC 1 workstations. The second PE was only 50% loaded in the dual PE optimized convolution case. This is due to not partitioning the long FIR filter for the channel response into two separate pipelined FIR filters, resulting in a computational load imbalance.

Partitioning the FIR over multiple PEs would be relatively easy, but for adaptive FIR (AFIR) filters it would be significantly more difficult. If deemed necessary to redistribute computations for optimal processor loading, an AFIR could be coded as standard start, middle, and end user callable library functions, with a run time selectable number of taps in each. The feedback of the filter error to previous blocks of the distributed filter can be delayed by one computational period without any significant effect. References [Long87a], [Long89a] study this problem and it is generally possible to delay the error

Figure 6.4 HDSL Transmitter-DFE Receiver Partitioned Over Two PEs

Table 6.2: Execution Time Comparison for the Simple HDSL Algorithm (2 PEs)

System	Execution Time (mS/Block)	Times Real-Time (R.T.=2.5 μ S/Block)	Improvement Ratio ^a	Processor Loading (%)
Sun 3/60	192.3	76,920	1.0	n.a.
Sun SPARC I	7.4	2,960	26 (1.0)	n.a.
Single PE (No optimization)	.673	269	286 (11.0)	100
Dual PE (No optimization)	.450	180	427 (16.4) <1.49>	100, 50
Single PE (Opt. convolution)	.159	64	1125 (46.5) <1.0>	100
Dual PE (Opt. convolution)	.103	41	1867 (72) <1.54>	100, 54.6

a. The first value is w.r.t. the Sun 360 workstation, the second value w.r.t. the Sun SPARC I, and the third w.r.t. single PE case.

feedback up to eight tap delays without detriment to the final result. This eliminates any latency effects that would occur if the error had to be fed back to previous stages within the same baud interval. It should also be observed that when splitting an AFIR filter, the intercommunication requirements increase from 1 to 3 data items between blocks (data, sum, and error). The additional intercommunication time is generally small with respect to the computational period, due to the fact that the filter is usually only split when a large number of taps is being computed, implying a lengthy computational period. As previously noted, the filter in this example was not split up as it was deemed sufficient for purposes of demonstration to observe the execution times, intercommunication time, and processor loading.

Figure 6.5 Fully adaptive HDSL Transmitter-DFE Receiver Partitioning (4 PEs)

The second HDSL example is a fully adaptive HDSL data transmitter/channel/DFE receiver simulation, partitioned onto four PEs as illustrated in figure 6.4. The receiver consists of an adaptive echo canceller, adaptive feed forward filter, and an adaptive decision feedback equalizer. The execution times listed in Table 6.3 are for the fully functional phase of operation after the initial learning sequences for the adaptive elements have occurred. The computational loading of the PEs is again unbalanced in this example, and is the limiting factor for the achievable speedup. To distribute the computation more evenly it would be necessary to split the longest AFIR filter over multiple PEs, as was mentioned for the previous example. As would be expected from the computational com-

plexity and the number of filter taps, the echo canceller subblock has the longest execution time and is the pacing element in the speedup possible (without partitioning it over multiple PEs).

Table 6.3: Execution Time Comparison for the Full HDSL Algorithm (4 PEs)

System	Execution Time (mS/Block)	Times Real-Time (R.T.=2.5 μ S/Block)	Improvement Ratio ^a
33 MHz 386/387	35.93	14,360	1.0
Single PE (No optimization)	3.40	1,360	10.6
Single PE (Opt. convolution)	1.80	720	19.9 (1.0)
Quad PE ^b (Opt. convolution)	.55	220	65.3 (3.3)

a. The first value is w.r.t. the workstation, the second value w.r.t. the single PE case.

b. Computational loading:

PE1	39%
PE2	54.5%
PE3	100%
PE4	59%

Total loading: 1.39 mS/Symbol (63%)

Communication Overhead: .103 mS/Symbol

R/C ratio: 17.51

6.2.4 Other Execution Examples

Oakley [Oak91a] has implemented image processing algorithms on this system. The image is partitioned into blocks by the designated master PE and distributed to the slave PEs. Each slave PE processes the data block and send the resulting block back to the master. The data intercommunication is performed concurrently to the block processing by using the C30 DMA controller and data buffering techniques. There is a communication overhead associated with maintaining the DMA server operation that is relatively independent of the number of processors being utilized or the block size being used (because the actual communication is done in the background and only the DMA server overhead consumes computational resources). The observed communication overhead was in the order of 20.5–23.5% for small block sizes (32) and 3.3–4.2% for large block sizes (256)¹. The overhead without the concurrent communication is 47–81% and 8.3–

1. Table 8.1 of [Oak91a].

12.8% respectively. For smaller block sizes the relatively longer communication times translates to a smaller R/C ratio. The wider variation is for varying number of PEs, because a larger number of PEs causes a greater percentage of intercommunication time with respect to the computational period. This illustrates that the concurrent communication can afford significant advantages, especially when the R/C ratio is low.

6.3 Conclusion

The method of hardware verification was reviewed. The multiprocessor benchmark results were presented for two example applications. Issues in porting these examples from a workstation to the multiprocessor were discussed. The execution times and relative speedup with respect to workstations were presented. It was noted that the major reason that the maximum speedup for these large grain problems was not achieved was due to the computational load imbalances, i.e. the difficulty in partitioning large subblocks evenly among processors. An independently performed application example (image processing) for which the load balancing was better distributed was discussed. In this case data intercommunication time was the limiting factor, and overlapped computation/communication was utilized for a significant speedup factor improvement.

Chapter 7

Conclusions

7.1 Conclusions

This thesis proposed a high performance slotted ring bus multiprocessor bus architecture that is suitable for simulating large-grain data flow DSP oriented system algorithms. The ground work for the proposal was based on a literature review of three areas. A study of the data flow of (DSP) algorithms was undertaken to characterize the algorithm subblock intercommunication characteristics. General bus architectures commonly used in multiprocessor systems were then reviewed. Next bus architectures used in commercial and research multiprocessor systems designed for DSP algorithm execution were reviewed. Furthermore, data flow characteristics of large-grain DSP communications algorithms of interest were briefly reviewed and compared to the characteristics described earlier.

Next, the suitability of the various previously reviewed multiprocessor intercommunication architectures to efficiently implement the stated data flow characteristics was considered. A proposed bus architecture that improves upon the deficiencies found was described. Then details of the multiprocessor hardware design were given, including the selection of a current DSP processor IC, the design and performance parameters of the bus architecture, and additional design details of the PE

A working system consisting of four processing elements was constructed and tested. Two representative algorithms were coded onto the system and benchmarked. The benchmarks indicate that a significant speedup in processing can be achieved over common workstation environments. It was noted that due to the large-grain nature of the algorithms, it was impossible to partition them onto the multiprocessor so that the computational loads were balanced. This results in a less than optimal speedup in the execution time of the algorithms.

This thesis has presented the specification, design, and testing of a DSP oriented

multiprocessor testbed for the modelling of communications design. This was accomplished with relatively limited resources. The design is of low cost and easily replicable. It is felt that the performance is significantly better than systems existing at the time of its inception, and is still competitive today.

7.2 Future Work

There are a number of areas that can be followed up on in the future. The bus architecture concept is technology independent. This allows maintenance of the processor by upgrading to the latest technology when there becomes a significant performance gap. Since the C30 and bus I/O processor are well partitioned at the hardware level, this can be accomplished independently (at the hardware design level) for each portion.

The multiprocessor can be linked with higher level DSP multiprocessor system design tools, such as Comdisco's SPW. In fact, this has already been accomplished at Carleton under the auspices of a separate project. The aforementioned product recently introduced the MPX multiprocessor interface, and this was successfully integrated onto this multiprocessor by D. Withey.

The investigation of finer-grain partitioning of the subblocks could be performed to allow more even computational load distribution. This would allow speed up ratios closer to ideal. Algorithm subblocks that are good candidates for intra-block partitioning implementation are the heavily used (in many communication DSP systems) FIR and AFIR filters, and the FFT.

References

-
Albe85a	G.Albert, D.Fronczak, J.McKinney, "High throughput signal processing system", <i>IEEE ICASSP</i> , August 1985, pp. 1621-1623.
Atla90a	Atlanta Signal Processors Inc., "Banshee System Instruction Manual - Release 1.3", Atlanta Signal Processors Inc., 1990, Part No. BAN-001-11/90.
Atla91a	Atlanta Signal Processors Inc., "MPI Multiprocessor Interface Board Instruction Manual - Version 1.35", Atlanta Signal Processors Inc., 1991, Part No. MPI-001-05/91.
Bhuy89a	Laxmi N. Bhuyan, Qing Yang, Dharma P. Agrawal, "Performance of Multiprocessor Interconnection Networks", <i>IEEE Computer</i> , February 1989
Bier90a	Jeffrey C. Bier, S. Sriram, Edward A. Lee, "A Class of Multiprocessor Architectures for Real-Time DSP", <i>IEEE VLSI Signal Processing IV</i> , IEEE Press, 1990, pp 295.
Bolc87a	Gunter Bolch, <i>et al.</i> , "MUPSI: A Multiprocess for Signal Processing", <i>Proceeding of the IEEE</i> , Volume 75, No. 9, September 1987.
Cast88a	Guido Castellini, Pierluigi Emiliani, Laura Pierucci, Franco Pirri, Santina Rocchi, "A Multiprocessor Structure Based on Commercial DSP", <i>IEEE ICASSP 1988</i> , pp 2108-2111.
Chin90a	J.W.Chinneck, R.A.Goubran, G.M.Karam, M.Lavoie, "A Design Approach for Real-time Multiprocessor DSP Applications", <i>Report SCE-90-05</i> , Real-Time and Distributed Systems Group, Dept. of Systems and Computer Engineering, Carleton University, February 7 1990.
Darl88a	P. Darlington, S.J. Elloit, "Synchronous Adaptive Filters with Delayed Coefficient Adaption", <i>IEEE ICASSP</i> , 1988, pp. 2586-2589.
Davi80	S. Davis, P. Mermelstien, "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences", <i>IEEE Transactions of Acoustics, Speech, and Signal Processing</i> , Volume ASSP-28, 1980, pp. 357-366
-

-
Deca89a	J. Decaluwe, J. Rabaey, J. Meerbergen, H. DeMan, "Interprocessor Communication in Synchronous Multiprocessor Digital Signal Processing Chips", <i>IEEE Transactions of Acoustics, Speech, and Signal Processing</i> , Volume 37, No. 11, December 1989, pp. 1816-1827.
Falc82a	D.D. Falconer, "Adaptive Reference Echo Cancellation", <i>IEEE Transactions on Communications</i> , Volume COMM-30, No. 9, September 1982, pp. 2083-2094.
Gaud85a	Jean-Luc Gaudiot, Rex Vedder, George Tucker, Dennis Finn, Michael Campbell, "A Distributed VLSI Architecture for Efficient Signal and Data Processing", <i>IEEE Transactions on Computer</i> , Volume C-34, No. 12, December 1985.
Gass86a	W. Gass, R. Tarrant, G. Doddington, "A parallel signal processing system", <i>IEEE ICASSP</i> , April 1986, pp. 2887-2890.
Gass87a	W.S. Gass et al., "Multiple digital signal processor environment for intelligent signal processing", <i>Proceedings of the IEEE</i> , Vol. 75, No. 9, September 1987.
Good81a	James R. Goodman, Carlo H. Seequin, "Hypertree: A Multiprocessor Interconnection Topology", <i>IEEE Transactions on Computers</i> , Vol. C-30, No. 12, December 1981.
Haim90a	R. Haimi-Cohen, H. Herzberg, Y. Be'ery, "Delayed Adaptive LMS Filtering: Current Results", <i>IEEE ICASSP</i> , 1990.
Haye88a	John P. Hayes, "Computer Architecture and Organization", 2nd Edition, McGraw-Hill Book Company, 1988, ISBN 0-07-027366-9.
Hoan90a	P. Hoang, J. Rabaey, "Program Partitioning for a Reconfigurable Multiprocessor System", <i>IEEE VLSI Signal Processing IV</i> , IEEE Press, 1990, pp. 53-62.
IDT90a	Integrated Device Technology Inc., "High-Speed Tri-Port Bus Multiplexer", <i>Data sheet</i> , Integrated Device Technology Inc., October 1990, publication num. DSC-4627/1.
Kaba87a	P. Kabal, "The Stability of Adaptive Minimum Mean Square Error Equalizers Using Delayed Adjustment", <i>IEEE Transactions on Communications</i> , Vol. Com-31, No. 3, March 1983, pp. 430-432.
-

Kahr88a	M. Kahr, "The Architecture of DSP*: A DSP Multiprocessor", <i>IEEE ICASSP</i> , 1988.
Kalo87a	N. Kalouptsidis, S. Theodoridis, "Parallel Implementation of Efficient LS Algorithms for Filtering and Prediction", <i>IEEE Transactions on Acoustics, Speech, and Signal Processing</i> , Volume ASSP-35, No. 11, November 1987.
Koh88a	W. Koh, A. Yeung, P. Hoang, J. Rabaey, "A Multiprocessor System for DSP Behavioural Simulation", <i>VLSI Signal Processing III</i> , IEEE Press, 1988, pp.295-306.
Kung82a	H.T.Kung, "Why Systolic Arrays", <i>IEEE Computer</i> , January 1982, pp.
Lawr83a	V.B.Lawrence, S.K.Tewksbury, "Multiprocessor implementation of adaptive digital filters", <i>IEEE Trans. on Communications</i> , Vol. COM-31, No. 6, June 1983, pp. 826-835.
Lee87a	Edward A. Lee, David G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", <i>IEEE Transactions on Computers</i> , Volume C-36, No. 1, January 1987.
Lee87b	Edward A. Lee, David G. Messerschmitt, "Pipeline Interleaved Programmable DSP's: Synchronous Data Flow Programming", <i>IEEE Transactions on Acoustics, Speech, and Signal Processing</i> , Volume ASSP-35, No. 9, September 1987.
Lee87c	Edward A Lee, David G. Messerschmitt, "Synchronous Data Flow", <i>Transactions of the IEEE</i> , Volume 75, No. 9, September 1987.
Lee88a	Edward A. Lee, "Programmable DSP Architectures, Part I", Submitted to <i>IEEE Transactions on Acoustics, Speech, and Signal Processing</i> , July 14, 1988.
Lee88b	Edward A Lee, "Programmable DSP Architectures, Part II", Submitted to <i>IEEE Transactions on Acoustics, Speech, and Signal Processing</i> , July 14, 1988.
Lee89a	Edward A Lee, Wai-Hung Ho, et. al., "Gabriel: A Design Environment for DSP", <i>IEEE Transactions on Acoustics, Speech, and Signal Processing</i> , Volume ASSP-37, No. 11, November 1989, pp. 1751-1762.

-
Lin90a	D.W.Lin., "Minimum Mean-Squared Error Echo Cancellation and Equalization for Digital Subscriber Line Transmission: Part I-Theory and Computaion", <i>IEEE Transactions on Communications</i> , Volume 38, No. 1, January 1990, pp. 31-38.
Lin90b	D.W.Lin., "Minimum Mean-Squared Error Echo Cancellation and Equalization for Digital Subscriber Line Transmission: Part II-A Simulation Study", <i>IEEE Transactions on Communications</i> , Volume 38, No. 1, January 1990, pp. 39-45.
Long87a	G. Long, F. Ling, J. Proakis, "Adaptive Transversal Filters with Delayed Coefficient Adaptation", <i>IEEE ICASSP</i> , 1987, pp. 431-433.
Long89a	G. Long, F. Ling, J. Proakis, "The LMS Algorithm with Delayed Coefficient Adaptation", <i>IEEE Trans. on Acoustics, Speech and Signal Processing</i> , Vol. 37, No. 9, September 1989, pp. 1397-1405.
Meye88a	M.D.Meyer, D.P.Agrawal, "Adaptive Lattice Filter Implementation in Pipelined Multiprocessor Architectures", <i>IEEE ICASSP</i> , September 1988, pp. 1580-1583.
Mill85a	T.K.Miller, S.T.Alexander, L.J.Faber, "An SIMD Multiprocessor Ring Architecture for the LMS Adaptive Algorithm", <i>IEEE Trans. on Communications</i> , Vol. COMM-34, No. 1, January 1986, pp. 89-92.
Mill85b	T.K.Miller, S.T.Alexander, L.J.Faber, "An implementation of the LMS Adaptive Filter Using an SIMD Multiprocessor Ring Architecture " <i>IEEE ICASSP</i> , August 1985, pp. 1605-1608.
Moha90a	P. Mohanraj, D.D.Falconer, T.A.Kwasniewski, "Baseband Trellis Coded Modulation With Combined Equalization/Decoding for High Bit Rate Digital Subscriber Loops ", <i>Globecom</i> , 1990, pp. .
Morg85a	D.P.Morgan, H.F.Silverman, "An investigation into the efficiency of a parallel TMS320 architecture: DFT and speech filterbank applications", <i>IEEE ICASSP</i> , August 1985, pp. 1601-1604.
Morg90a	N. Morgan, J. Beck, E. Allman, and J. Beer, "RAP: A Ring Array Processor for Multilayer Perception Applications", <i>IEEE ICASSP</i> , 1990,
Muel79a	.K.H.Mueller "Combining Echo Cancellation and Decision Feedback Equalization", <i>Bell Systems Technical Journal</i> , Vol. 58, No. 2, February 1979, pp. 491-500.
-

Oakl91a	D.Bruce Oakley, "Image Filtering on Slotted Ring Multiprocessor DSP Systems", <i>Thesis at Systems and Computer Engineering</i> , Carleton University, Ont., Canada, 1991.
OSha90a	D. O'Shaughnessy, "Speech Communication", Addison-Wesley Publishing Company, 1990, ISBN 0-201-16520-1.
Pars87a	T.W. Parsons, "Voice and Speech Processing", McGraw-Hill Book Company, 1987, ISBN 0-07-048541-0.
Potk90a	M.Potkonjak, J. Rabaey, "Retiming for Scheduling", <i>IEEE VLSI Signal Processing IV</i> , IEEE Press, 1990, pp 23-32.
Reed83a	Daniel A. Reed , and Herbert D. Schwetman, "Cost-Performance Bounds for Multimicrocomputer Networks", <i>IEEE Transactions on Computers</i> , Volume C-32, No. 1, January 1983.
Rotg88a	Angel C. Rotger-Mora, "An N-Dimensional SIMD Ring Architecture for Implementing Very Large Order Adaptive Digital Filters", <i>IEEE ICASSP</i> , 1988.
Sant88a	J.Santos, J.Parera, M.Viega, "A Hypercube Multiprocessor for Digital Signal Processing Algorithm Research", <i>IEEE ICASSP</i> , September 1988, pp. 1698-1701.
Ston87a	Harold S.Stone, "High-Performance Computer Architecture", Addison-Wesley Publishing Company, 1987, ISBN 0-201-16802-2.
Texa86a	Texas Instruments Inc., "SN54AS8839/ SN74AS8839 32-Bit Shuffle Exchange Network", <i>Data sheet</i> , Texas Instruments Inc., June 1986, Part No. SDAS100.
Texa86b	Texas Instruments Inc., "SN74AS8840 Digital Crossbar Switch", <i>Product preview</i> , Texas Instruments Inc., 1986, Part No. SDBV010.
Texa88a	Texas Instruments Inc., "Third-Generation TMS320 User's Guide", Texas Instruments Inc., August 1988, Part No. SPRU031.
Texa90b	Texas Instruments Inc., "TMS320C3x User's Guide", Texas Instruments Inc., April 1990, Part No. SPRU031a.
Theo90a	Sergios Theodoridis, "Pipeline Architecture for Block Adaptive LS FIR Filtering and Prediction", <i>IEEE Transactions on Acoustics, Speech, and Signal Processing</i> , Volume 38, No. 1, January 1990, pp. 81-90.

Theo90b	S. Theodoridis, N. Kalouptsidis, D. Bakirtzis, "Pipelined Algorithm for LS FIR Filters with Symmetric Impulse Response", <i>IEEE Transactions of Acoustics, Speech, and Signal Processing</i> , Volume 38, No. 2, February 1990.
Vita87a	Vita, "The VMEbus Specification", VMEbus International Trade Association, 1987.

END

15-1 2-93

FIN