1

of/de

1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

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

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

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

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

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

AVIS

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

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

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

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

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

**Hardware and Software for
A Speech and Signal Processing Subsystem
for a Multiprocessor**

by

© Craig J. McLurg -

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

Department of Systems and Computer Engineering,
Carleton University,
Ottawa, Ontario, Canada.

April, 1987

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

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

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

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

ISBN 0-315-39425-0

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

*Hardware and Software for
A Speech and Signal Processing Subsystem
for a Multiprocessor*

- submitted by Craig J. McLurg, BSc., in partial fulfilment of the requirements for the degree of Master of Engineering.

Thesis Supervisor

Chairman,
Department of Systems and
Computer Engineering

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
April, 1987

Abstract

This thesis describes a powerful, programmable digital signal processing subsystem which is integrated into a multiprocessor to provide digital signal processing capabilities and a simple interface to these capabilities. The signal processing subsystem has a faster execution speed and specialized instructions which, when integrated into the multiprocessor system, allow high performance applications such as speech and signal processing to be performed. In this case, speaker-dependent independent word speech recognition is performed. The signal processing server (SPS) is a collection of tasks which provides the interface between the multiprocessor system and the signal processing subsystem. A processor communication channel (PCC) was developed to allow the two processors in the subsystem to communicate. The design of the signal processing subsystem, including the SPS and PCC, is discussed and the speech recognition algorithms are described.

Acknowledgements

I would like to thank my supervisor, Professor C. M. Woodside, for all his help and encouragement.

I would also like to thank Kim Rowe of the ARTT Lab, for the ideas and assistance he provided during the project.

Contents

1	Introduction	1
2	Background	6
2.1	Multiprocessing	6
2.2	Harmony Operating System	10
2.3	Server Concepts	14
2.4	Justification of a Dedicated Signal Processor for Speech Recognition	15
3	System Design	21
3.1	Requirements	21
3.2	Signal Processing Subsystem Hardware Design	24
3.3	System Software Structure	30
3.4	Analog I/O Board	34
4	Processor Communication Channel	37
4.1	Communication Requirements	37
4.2	User Interface	38
4.3	Data Transfer	39
4.4	Mailboxes Via Shared Memory	41
4.5	FIFO Buffers	44

4.6	Final Implementation	46
5	Server Task	48
5.1	Server Design	49
6	Application to Speech Processing	59
6.1	Speech Recognition Today	59
6.2	Speech Recognition Algorithms	61
6.2.1	Feature Extraction	63
6.2.2	Pattern Similarity Detection	70
6.2.3	Decision	74
6.3	Implementation and Testing	74
7	Conclusion	77

List of Figures

2.1	Cross-bar Switch Network.	7
2.2	Multiport Memory Network.	8
2.3	Common Bus Network.	8
2.4	Harmony Multiprocessor System.	9
2.5	Message Passing Execution Timing.	13
2.6	Task Diagram.	16
2.7	Summary of Execution Times in msec.	19
3.1	First Design - Using Global Memory.	26
3.2	Second Design - All Possible Memory Locations.	29
3.3	Final Design.	31
3.4	System Software Structure.	32
3.5	Analog I/O Board Block Diagram.	35
5.1	Single-User Prototype System.	51
5.2	Multiple Signal Processing Subsystems.	53
5.3	Optimized Single User System.	55
5.4	Optimized Multi-User System.	57
6.1	Speech Recognition System.	62

6.2	Dynamic Time Warping Function.	72
6.3	Comparison of Execution Times per Frame in msec.	75

Chapter 1

Introduction

The processing capabilities of a general-purpose multiprocessor can be enhanced with the addition of a digital signal processing subsystem. Applications such as robotics and control have processing requirements which are satisfied by a general-purpose multiprocessor. But other applications, such as speech and signal processing, have strict processing requirements that require greater processing power than a general-purpose multiprocessor can provide. The signal processing subsystem has a faster execution speed and specialized instructions which, when integrated into the multiprocessor system, allow high performance applications to be performed while maintaining the advantages of multiprocessing such as reliability and modularity. Challenges facing designers are to integrate these signal processing capabilities into an existing multiprocessor system, and to provide future system users with an quick and easy way to use the system. This is particularly applicable in a laboratory environment.

This thesis describes a powerful, programmable digital signal processing subsystem which is integrated into a multiprocessor to provide digital signal processing capabilities and a simple interface. The aims of this integration are to provide digital signal processing capabilities which can be utilized by the entire multipro-

cessor system and to provide a simple interface to allow other tasks in the system to utilize these same signal processing capabilities. The interface is designed to be simple to use and understand so that system designers will have no trouble making use of the interface when writing application programs. These signal processing capabilities are used to implement a speaker-dependent independent word speech recognition system.

A signal processing subsystem, consisting of a specialized digital signal processor, general-purpose microprocessor, and analog I/O capabilities, has been constructed. This signal processing subsystem will provide the necessary digital signal processing capabilities to the general-purpose multiprocessor system once they are integrated together. The high-speed digital signal processing capabilities are provided by the dedicated signal processor, a Texas Instruments TMS32020 [23]. To provide a gateway to the rest of the system, a general-purpose processor, the Motorola MC68020 [22], is integrated into the Harmony multiprocessor system [8]. A signal processing server (SPS), a set of tasks which runs on the general-purpose processor, is utilized to interface the signal processing subsystem to the rest of the multiprocessor system. User tasks can then communicate with the signal processor using the existing message passing mechanism provided by the Harmony multiprocessor, multitasking operating system [8].

The features of the solution of adding digital signal processing capabilities to a general-purpose multiprocessor are the digital signal processor, the processor communication channel (PCC), and the signal processing server (SPS). Digital signal processors [13] are different from general-purpose microprocessors because they are essentially reduced-instruction-set computers optimized to execute digi-

tal signal processing algorithms. That is, they are optimized to perform integer addition, subtraction, multiplication, and shifting operations. Usually a barrel shifter and array multiplier are included to allow single-cycle execution of these functions. Since the most frequent operations used in digital signal processing are multiply and accumulate or subtract, digital signal processors provide very good performance. The PCC provides a link between the general-purpose microprocessor and the digital signal processor through which data and commands are passed. In addition, data integrity is guaranteed through mutually exclusive access to the data, and entire messages are passed between the two processors in an effort to be consistent with the communication primitives provided by the Harmony operating system. In addition to providing an interface to the signal processing subsystem for the user tasks, the SPS executes the protocol for using the PCC and controls the tasks which perform the actual reading and writing of data to the PCC.

Other advantages of digital signal processors are that they have approximately the same performance as the array processors used ten years ago to perform digital signal processing, but they are smaller, less expensive, and use less power by a factor of a thousand than their predecessors. These factors will encourage the widespread use of digital signal processors.

Chapter 2 provides background information on several topics which are utilized in this thesis. They are multiprocessing, the Harmony operating system, the server construct, and the need for a digital signal processor to perform speech recognition.

Chapter 3 describes the hardware and software system design of the Signal Processing Server (SPS).

In Chapter 4, the design of the Processor Communication Channel (PCC) is

discussed in detail. The PCC allows the transfer of data and instructions between the dedicated signal processor and the general purpose microprocessor which comprise the dedicated signal processing subsystem.

Chapter 5 describes the server task design and functionality. The purpose of the server task is to provide an interface for the signal processing subsystem to the rest of the multiprocessor system.

In Chapter 6, a brief summary of the state of speech recognition today is presented, and the speech recognition algorithms and their implementation is discussed.

The conclusions for the thesis are stated in Chapter 7.

The author's research included:

- Defining the structure of the server and designing and implementing all of the server software (this includes the user interface task, multiprocessor interface tasks, processor communication tasks, and signal processor interface routines).
- Designing and debugging the processor communication channel hardware so that the two processors can exchange messages.
- Implementing the processor communication routines on the signal processor.
- Implementing speech recognition routines on the signal processor.
- Participating in the initial design stages of the server hardware.
- Designing and debugging an analog I/O board with 12 bits of precision which interfaces to the TMS32020.

The remainder of the server hardware development was the responsibility of another student [9]. This hardware included the signal processor and general-purpose microprocessor circuits.

Chapter 2

Background

This chapter provides background information on several topics which are utilized in this thesis. They are multiprocessing, the Harmony operating system, the server construct, and the need for a digital signal processor to perform speech recognition.

2.1 Multiprocessing

A multiprocessor system consists of a group of processors working together toward a common aim. The reasons for grouping processors together are speed and reliability. A system's performance can be increased by having several processors performing operations in parallel. Ideally n processors connected together will provide an n -fold increase in performance over a single processor. Unfortunately this isn't true because of the overhead involved in coordination and communication between the processors. The reliability of the multiprocessor system can be improved by having the remaining processors distribute the workload of a failed processor. Reliability and fault tolerance must be designed into a system from the beginning and a performance sacrifice is incurred due to overhead.

The three most common types of multiprocessor interconnection strategies [1] are

Figure 2.1: Cross-bar Switch Network.

- Cross-bar Switch (figure 2.1).
- Multiport Memory (figure 2.2).
- Common Bus (figure 2.3).

A method of interconnecting the processors, which execute instructions and manipulate data, and memory modules, which store instructions and data, is necessary to allow the movement of data and instructions. The cross-bar switch is the most extensive connection scheme providing a direct path from each processor to each memory module. However, as the number of processors and memory modules increases, the number of crosspoints increases by a factor of n^2 until the switching network becomes the most expensive part of the system. The multiported memory scheme has a path from each processor to each memory module and access arbitration is done at the memory module. The common bus organization is the simplest although it can become more complex if more than one bus is used. The

Figure 2.2: Multiport Memory Network.

Figure 2.3: Common Bus Network.

Figure 2.4: Harmony Multiprocessor System.

major limitation with the common bus architecture is that only one processor may access the bus at once. Bus contention occurs if a processor tries to access the bus while it is in use. The processor trying to access the bus will have to wait until the other has finished its access.

The Harmony multiprocessor system (figure 2.4) is configured as a common bus but each processor has memory associated with it. All of the memory in the system is configured as a globally-accessible linear address space so other processors can access a particular processor's memory. Communication is based upon message passing and interprocessor interrupts are used to notify processors of messages.

2.2 Harmony Operating System

The Harmony operating system [8] is a multitasking, multiprocessing operating system for real-time control. It is a portable system in that it can be used on many different hardware configurations. The only constraints on the hardware is that it must provide for interprocessor interrupts and support a linear address space. The memory in the multiprocessor system is organized as a linear address space which is shared by all the processors. It is straightforward to integrate a new device into both the physical level of connection and the higher levels of software. Harmony does not support program development and consequently executable code is cross-compiled and downloaded from a host computer. Tasks are the basic executable unit and Harmony supports multiple tasks on each processor. The Harmony operating system executes on a multiprocessor system which is based upon the VMEbus. Up to 16 processor cards may be included in a system configuration.

Message passing systems may be organized in several ways [4], the simplest of which involves the two primitives, Send and Receive. The send primitive transmits a message to the receiver and continues execution. When a receive primitive is executed, the task halts execution, or is blocked, until it receives a message. It will wait indefinitely to receive a message. If several messages are queued, then the first message in the receive queue is taken when a receive primitive is executed. The problems with these simple primitives are that no notification is received when a message is sent, there is no explicit task synchronization, and a series of uncontrolled sends could overflow a task's receive queue.

In order to provide notification of a message being received and task synchro-

nization, two new primitives are introduced to be used instead of Send and Receive: Send.Wait and Receive.Reply. When a task calls Send.Wait, a message is sent and the task doesn't continue execution, it is blocked until it receives a reply message from the receiver task. The receiver task executes the Receive.Reply primitive and when a message is received, a reply or acknowledge is sent back immediately. Thus explicit task synchronization is provided along with notification of message reception.

The message passing primitives Send.Wait, Receive, and Reply are used to allow for easier transfer of data between a requesting task and a utility task. The requesting task passes data to a utility task who modifies it and returns the data to the requesting task. If the primitives Send.Wait and Receive.Reply are used, two separate calls to each must be made: first to pass the data to the utility task, and second, to return the modified data to the requesting task. Of the new primitives, Send.Wait remains the same, a message is sent and the sender remains blocked until a reply is received. Thus notification of message delivery and task synchronization is preserved. However, the receiving task executes a Receive to accept messages and it executes a separate primitive, Reply, to send a reply message and unblock the sending task. In between executing a Receive and a Reply, the receiving task may perform other execution, such as modifying the data, and then pass the result back to the sending task with the reply message. The sequence Send.Wait, Receive, and Reply can be thought of as corresponding to the extended rendezvous as used in Ada [4]. However, unlike the Ada rendezvous [3], where only nested rendezvous are accepted, a receiving task may receive and reply to tasks in any order. This is useful for keeping tasks blocked such as a

controlling task blocking worker tasks until work is available.

The Harmony operating system is based on message passing. Messages usually consist of a message type and a message data field. The basic communication primitives supported are Send_Wait, Receive, and Reply. Send_Wait and Reply are task specific while Receive may be task specific or general, that is, receive a message from a specific task or else from anyone who sends to it. Another primitive, Try_Receive, is also supported. It is simply a non-blocking Receive. Try_Receive is used to accept a message if the receiver cannot be blocked. If a message is queued, it is accepted otherwise execution continues: the task does not block.

Figure 2.5 illustrates the different execution sequences which can occur during a Send_Wait, Receive, and Reply sequence. It is assumed that both tasks are running on different processors so that both tasks may execute at the same time. Figure 2.5a illustrates what occurs when the sender executes a Send_Wait before the receiver executes a Receive. The sender is blocked until a Reply is received from the receiver and the sent message is queued until a Receive is executed by the receiving task. Between receiving a message and executing a Reply, the receiver can perform data manipulations and send the modified data back to the sending task. If the receiver executes a Receive before the sender sends a message, as shown in figure 2.5b, then the receiver becomes blocked until a message is sent. Once the sender sends its message, it becomes blocked until a reply is received from the receiver but the receiver receives the message and may begin to process the data. When the receiver executes a Reply, both tasks continue execution. Note that when a Reply is executed, both tasks are synchronized, that is, they are both

Figure 2.5: Message Passing Execution Timing.

at a known point in their execution with respect to each other.

2.3 Server Concepts

Within the Harmony operating system, the server construct [8] is used to create intelligent peripherals. The server acts as a controlling task to provide the level of interface desired rather than that which the hardware provides. A user task can use the server simply by sending it a message. The server task can provide a very sophisticated interface, particularly if worker tasks are used by the server. The server can accept data and pass the data to a worker task. The worker task can format the data and interact with a peripheral using a particular protocol. If a server task is used to interact with a peripheral, the user task has a simpler data structure and a cleaner interface than if the user directly interacted with the peripheral. The server task hides the existence of the worker tasks from the user tasks. The use of server tasks also provides other benefits. Mutual exclusion can be enforced since the server can control access to the resource. A high degree of parallelism is possible because the calling task can perform other work while the server is processing the request if a transport task was used to make the request. The server guarantees completion of the request and holds the results until the user task is ready to pick them up.

The server can never become blocked on a Send.Wait call because it only accepts messages, it never sends them. Thus the server is only blocked when it has nothing to do. It is waiting to accept a request. If the server made calls to other tasks, it could be blocked pending acceptance by the destination task. This means that the server wouldn't be able to provide service to other tasks in the system.

By having the server only accept requests, the problem is eliminated. Plus, service is provided on a first-come first-served basis. As in figure 2.6a, the USER tasks call to make requests and the WORKER tasks call to obtain work. The USER tasks are blocked by the SERVER task until the request is complete. This provides process control and prevents a USER task from queuing up an enormous number of requests. The WORKER tasks are blocked when there is no work so when a request does arrive, the SERVER task simply replies to a WORKER task with the request and off he goes. The WORKER task makes another call to the SERVER task when finished and requests more work.

If two tasks, for instance the USER task and the SERVER task, need to communicate with each other but neither should be blocked by a Send/Wait call, then a transport task is used. A transport task would be used by the USER task to relay messages to the server (figure 2.6b). It calls the USER task to accept messages to be passed to the SERVER task, sends the message to the server, is blocked until a reply arrives, and returns the result back to the USER task. Thus neither the USER task nor the SERVER task is blocked waiting for the other.

2.4 Justification of a Dedicated Signal Processor for Speech Recognition

A dedicated signal processor will be used to perform speech recognition because it can execute the necessary algorithms faster than a general-purpose microprocessor. Quick execution of the speech recognition algorithms is necessary to provide real-time speech recognition, which is the objective for the completed system. The speech recognition algorithms are thoroughly discussed in chapter six.

A processor is needed to execute the speech recognition algorithms. The crite-

Figure 2.6: Task Diagram.

ria for choosing a processor are speed and ease of implementation. Three processor options are available:

- The general purpose microprocessors available in the existing system.
- A dedicated signal processor.
- A bit-slice design.

The speed of the processors can be compared by estimating the time it takes each processor to execute the speech recognition algorithms. These execution times can be estimated by coding the algorithm and looking up the execution times for each instruction. Only the algorithms themselves were coded, no attempt was made to write the connecting code. It is assumed that the processors would require the same amount of housekeeping processing. The execution times are estimated for both the signal processor and the general-purpose microprocessor but not for the bit-slice design. A bit-slice design can provide a further increase in speed of 50% over the TMS32020 [11]. But a bit-slice design would be an order of magnitude more complex than a single-chip processor design and a 50% increase in speed is not offset by such a leap in complexity. The MC68010 [21] and the TMS32020 [23] were selected as the processors to be used to estimate the execution times since they are the microprocessors currently available in the existing multiprocessor system, and the state of the art signal processor, at the time. Recently, Texas Instruments has introduced another signal processor, the TMS320C25 [24], which is upwardly compatible and approximately twice as fast.

The MC68010 has an 8 MHz clock giving a 125 ns cycle time. However the instructions are multicyle, taking from 4 cycles (register to register move) to

142 cycles (signed divide). The TMS32020 is a specialized VLSI digital signal processor. As such it has a tailored instruction set and could be considered a RISC machine. It has a 5 MHz clock with 200 ns cycle time. Most of its instructions are completed within a single clock cycle. This includes a single cycle multiply when used with data from internal memory. The single greatest advantage the signal processor has over the general-purpose microprocessor is multiplication speed. The signal processor can perform a 16-bit signed multiply in one cycle or 200 ns. The general-purpose microprocessor can perform a 16-bit signed multiply in 46 cycles or 5740 ns.

The instruction times for the TMS32020 differ depending upon whether the data is stored in on-chip or external memory. The TMS32020 has 544 words of internal data RAM and can address up to 64K words of external data RAM. Instructions normally take one cycle (200 ns) to execute with the instruction operand stored in the on-chip memory. However, if the instruction operand is stored in external memory, the instruction will take two cycles (400 ns) to execute rather than only one cycle (200ns). But, if the instruction is pipelined using the repeat instruction, the execution time drops to one cycle. In computing the estimated execution time, the data is assumed to be stored in external memory causing most instructions to take two cycles rather than one. This assumption is made because the on-chip memory is not large enough to hold all of the data at once and so data transfers would have to be made to keep the being manipulated in the on-chip memory. The final implementation includes a lot of external memory data manipulations because the on-chip memory is too small.

Figure 2.7 provides a summary of the estimated execution times for the various

Algorithm	TMS32020	MC68010	SPEEDUP
Endpoint Detection	0.05	0.9	17.8x
Preprocessing	0.84	3.9	4.6x
Autocorrelation Analysis	3.24	23.6	7.3x
LPC Analysis	0.20	0.8	4.2x
Feature Detection	4.33	29.2	6.8x
Pattern Similarity Detection	14.20	65.7	4.6x
Vocabulary (1 second delay)	70 words	15 words	

TMS32020 Overall Speed Advantage is 5.1 times.

Figure 2.7: Summary of Execution Times in msec.

parts of the speech recognition algorithm. The TMS32020 has an overall speed advantage by a factor of approximately 5. This is due to the specialized nature of the 32020's instruction set as well as its single-cycle instruction execution, particularly the multiply instruction. The advantage of the 32020's specialized instruction set is especially evident for endpoint detection and autocorrelation analysis where large numbers of samples are multiplied and accumulated. The vocabulary size is determined by the number of pattern similarity detection operations that can be completed in one second. One second is an arbitrary time interval which is felt would provide rapid enough response to satisfy a user. The size of the vocabulary provided by the signal processor is adequate but that provided by the microprocessor isn't even functional. Obviously the TMS32020 should be selected to perform speech recognition, since it executes the speech algorithms faster and allows a larger vocabulary.

The intention of the speech recognition system described in this thesis is to

accept single-user commands to be passed to an application program. Since the project is situated in a laboratory setting, the speech recognition software can be tuned to provide faster execution and a greater vocabulary.

Chapter 3

System Design

This chapter discusses the requirements for and describes the hardware and software system design for the signal processing subsystem. The major hardware design iterations are reviewed, the software structure for the signal processing subsystem is introduced, and the analog input/output (I/O) board is briefly described.

3.1 Requirements

The major requirements for the signal processing subsystem are :

- Digital signal processing capabilities.
- Integration of the multiprocessor and the signal processing subsystem.
- A simple user interface.

The digital signal processing capabilities are provided by a specialized digital signal processing chip, the TMS32020. The existing processors in the multiprocessor system are powerful, yet not fast enough to perform signal processing. However, a signal processor can perform simple number crunching much faster

than a general-purpose microprocessor. If a signal processor obtains the raw data, does the initial data reduction, and passes the results to the other processors in the system, then the system will be capable of higher performance. Also if the signal processor performs the I/O in its own I/O space, rather than across the multiprocessor system bus, and performs the initial data reduction before passing data to the other processors in the system then there will be less bus traffic.

The integration of the multiprocessor and the signal processing subsystem can be accomplished by having the signal processing subsystem interface to the VME-bus and run the Harmony operating system. This will allow all the processors in the multiprocessor system and the signal processing subsystem to share memory and pass messages back and forth.

The simple interface provides a way for Harmony tasks to communicate with and control the signal processing subsystem. Thus, when a designer wants to set up a particular application the design time will be greatly reduced since only the signal processing code and a Harmony task need be written. The user interface to the signal processor is to be based upon the server concept, which allows a sophisticated user interface to run on the general-purpose multiprocessor and a very simple mechanism to control the process running on the signal processor. This signal processing server will provide a standardized user interface and transparent communication mechanism for any task in the Harmony multiprocessor system which wishes to communicate with the signal processor.

One alternative to having a server is to port the Harmony operating system to the signal processor. That is, allow the signal processor to execute the operating system itself, then other tasks could communicate directly with the signal

processor. Unfortunately there are several problems with this idea. Harmony is written in a high level language (C) and there isn't a compiler available for the TMS32020. Harmony uses a linear address space which is as big as necessary (about one Mbyte) and the TMS32020 has only 64K words of address space. Extra address bits would have to be generated to allow the TMS32020 to address the entire Harmony address space. The Harmony operating system itself takes up about 16K words of memory, leaving the remainder for application programs and data; memory space is already at a premium. The TMS32020 is a high speed dedicated signal processor and its performance would certainly be compromised by the addition of an operating system. Lastly, is a multitasking operating system needed on this sort of processor? It might be useful for multirate signal processing but probably wouldn't be an advantage for filter implementations. Porting the Harmony operating system to the TMS32020 would involve a lot of effort which the results don't justify.

The user interface provides a mechanism to allow the transfer of data and commands between a user task or tasks and the signal processing subsystem. In the case of speech recognition, character strings are passed across the interface and represent words to be learned or words recognized. The commands passed across the interface range from download program and begin execution, which are explicit messages,

to user-defined commands, which can be encapsulated in data messages. Examples of user-defined commands in the speech recognition system are "Learn Word" and "Recognize Word". The commands are discussed in section 4.3, where the user interface to the signal processing server is described. The maximum message size

for the speech application is eight words: one word for each of the message type and length, and six words to store the 12 characters allotted for each word in the vocabulary. The message rate is governed by the speed of recognition, whenever a word is recognized a message will be generated. The message rate is expected to be approximately one message every two seconds. The ESM (Electronic Support Measures) [9] application has a greater amount of data transfer. The maximum message size is expected to be 40 words. The average number of messages is expected to be low, perhaps one every ten seconds. However, at startup four messages may have to be sent one after another; one message to initialize each data channel.

The hardware for the signal processing subsystem is described in section 3.2, the signal processing system software is described in section 3.3, and the analog I/O board is briefly discussed in section 3.4. A detailed description of the PCC is provided in Chapter 4.

3.2 Signal Processing Subsystem Hardware Design

This section discusses the hardware design of the signal processing subsystem. The signal processing subsystem can be thought of as a unit which interfaces to the rest of the Harmony multiprocessor system via the VMEbus, the same as all the other processors but also provides digital signal processing capabilities. The VMEbus connects the signal processing subsystem to the multiprocessor physically and a general-purpose microprocessor provides the functional connection. The general-purpose microprocessor runs the Harmony operating system, which provides facilities for message passing and interrupt handling on an interproces-

sor basis, and a Harmony task called the signal processing server (SPS) executes on the general-purpose microprocessor providing access to the signal processing capabilities for other tasks running in the system. Besides the general-purpose microprocessor and the signal processor, the hardware design must include:

- VMEbus access to allow the other processors in the system access to the general-purpose microprocessor's memory so that they can communicate using the Harmony operating system.
- Analog I/O capabilities which are necessary to obtain data samples and output results.
- Memory to store programs and data in.

Two factors influenced the hardware design: the minimization of data transfers between processors, and mutual exclusion and data integrity.

The signal processor and general-purpose microprocessor must communicate with each other to transfer commands and data. The amount of data transferred at once could be quite high, for example, 1024 words for an FFT. If the two processors communicated with each other using the system bus, then much of the available bus bandwidth could be used up during such large data transfers and the performance of the entire system could be degraded. If the two processors use a private bus to communicate with each other, then they won't use any of the system bus bandwidth.

The major design iterations for the signal processing subsystem are reviewed. Only the final design was actually prototyped. The initial design utilizes a globally accessible memory to store programs and data for both processors and is shown

Figure 3.1: First Design – Using Global Memory.

in figure 3.1. By using a triple-ported global memory, there is no need to transfer data between different memories and introduce copying overhead. Data won't have to be transferred between the microprocessor and the signal processor. Programs won't have to be downloaded to the signal processor; only the starting address in the global memory need be passed. The VMEbus block refers to the interface to the rest of the system via the VMEbus. All the other processors in the system can access the global memory through the VMEbus interface. The Analog I/O block provides analog-to-digital conversion of analog inputs and digital-to-analog conversion to generate analog outputs. The analog I/O block is placed in the I/O space of the signal processor so that all of the samples are privately passed from the analog I/O block to the signal processor. The switch and arbitration unit refers to the mechanism which provides equal access to the global memory and resolves all conflicts.

There are several disadvantages with the design using a global memory. The switch and arbitration unit must be very fast so that wait states aren't introduced into every memory access. Contention for the memory is an issue. The signal processor will be running high-speed real-time programs and it shouldn't wait for memory accesses. Mutual exclusion would be difficult to enforce efficiently. Since tasks from other processors can access the global memory via the VMEbus interface, data integrity can't be guaranteed. If the currently executing program on the signal processor were written over, the results would be disastrous. The signal processor's address space is only 64K words and the global memory must be larger than that to accommodate Harmony tasks and data. Another determining factor is cost. Because the TMS32020 is running real-time programs, memory accesses

must be completed within one clock cycle. To have memory fast enough to allow for single-cycle memory access and decode time in the switch and arbitration unit, high density static RAM must be used throughout the global memory. This type of memory is very expensive. The disadvantages of the global memory far outweigh the advantages.

Figure 3.2 shows another view of the system with all logical possible locations for memory excluding the global memory, which was rejected previously, and local memory for the general-purpose multiprocessor, which serves no useful purpose.

These possible memories are:

1. Harmony linear address space - dual-ported and the same as all the other boards in the system. It is accessible by the on-board processor as well as all the other processors in the system via the VMEbus interface so that the processors can communicate with each other.
2. Shared memory - dual ported between the microprocessor and the signal processor to allow communication and access to shared data. The existence of this memory is thoroughly discussed in chapter 4.
3. Signal processor internal memory - This memory is part of the processor and is configurable as either data memory or both program and data memory. In this case, it has been configured as data memory but is very small and more memory is needed to do speech processing.
4. Signal processor data memory - needed because the internal data memory isn't very large. There could be a tradeoff in size between items 2 and 4.

Figure 3.2: Second Design - All Possible Memory Locations.

5. Signal processor program memory – used to store programs. Downloading programs to this memory can be done by transferring through the shared memory or else by providing a path from the microprocessor to this memory.
6. Signal processor IO space – the analog I/O block can be memory mapped in here to take advantage of specialized instructions to allow the efficient transfer of data.

The discussion above shows that each of the memories described is needed in the system. The only exception is the shared memory (2). Rather than using shared memory to provide communication between the signal processor and the general-purpose microprocessor, a processor communication mechanism (PCC) utilizing FIFO buffers will be used to provide communication between the two processors. The PCC is described in detail in the next chapter. The final design is shown in figure 3.3. The entire signal processing subsystem was to be placed on a single board but unfortunately had to be placed on two separate boards due to space considerations. The prototype was built using wire-wrap but if a printed circuit layout was used, it would fit onto one board. The two boards built are tightly interconnected and do not communicate over the VMEbus, ribbon cable is used instead.

3.3 System Software Structure

The overall system software structure of a typical application is illustrated in figure 3.4. The Server task provides an interface so User tasks can access the signal processing subsystem. The Send320 task passes data and commands to the signal processor, while the Rcve320 task accepts data and commands. The

Chapter 3 System Design

Figure 3.3: Final Design.

Figure 3.4: System Software Structure.

Send68 and Rcv68 routines perform analogous functions for the signal processor. These tasks comprise the signal processing server (SPS) and remain the same for all applications. To implement a new application, a designer need only write the User task and the signal processing routines.

The SPS provides an interface to the rest of the multiprocessor system for the signal processor. The SPS's purpose is twofold. It is designed to provide transparent communication to the signal processor, and, equally important, it also provides a user interface which greatly shortens the time to get an application up and running. The SPS allows the USER task to send messages to or receive messages from the signal processor, download and execute application programs, and terminate currently executing programs. The design of the SPS is based upon the administrator concept [7] with the SERVER task performing the role of the administrator. Its worker tasks are the Send320 task and the Rcv320 task which are used to write in and read out messages to and from the FIFO buffers. On the signal processor side, the Send68 and Rcv68 routines write and read messages to and from the FIFO buffers. Send68 is called as a subroutine while Rcv68 is an interrupt service routine. In all instances, interrupts are used for signaling rather than busy waiting.

To the USER task, the SPS provides an interface which allows the USER task to send and receive only entire messages. That is, the SPS provides message passing at the User interface level. The actual amounts of data that are passed through the PCC may be single words, message fragments, or entire messages. Thus the PCC can be viewed as having two levels of abstraction: the User Interface Level which provides message-based communication between the SPS and the

USER, and the Data Transfer Level which provides the actual transfer of data between the two processors. The PCC is discussed in detail in Chapter 4.

3.4 Analog I/O Board

The analog I/O board (shown in figure 3.5) consists of an input section and an output section, both with 12 bits of precision. The input section consists of:

- 4 channel analog multiplexor,
- programmable amplifier,
- sample and hold,
- 12 bit analog-to-digital converter.

The output section is comprised of:

- 12 bit digital-to-analog converter,
- 4 channel analog multiplexor.

In addition, for speech data, an optional codec filter is used to bandlimit the input and output samples.

An analog-to-digital converter was used rather than a codec because it was felt that an analog-to-digital converter would be easier to apply in applications other than speech and be generally more flexible. The analog I/O board is controlled by accessing ports in the I/O space of the signal processor. There are locations to read data from the analog-to-digital converter, write to the digital-to-analog converter, control the start and stop of sampling, and a data register. The data register is used to control the setting of the analog multiplexors, the gain of the

Figure 3.5: Analog I/O Board Block Diagram.

programmable amplifier, and set the sampling rate. The sampling rate is determined by limiting the number of counts that an 8-bit counter can have before it overflows setting an interrupt. A 256 kHz clock rate is used. A sampling rate of 8 kHz is used for speech.

Chapter 4

Processor Communication Channel

Communication is needed between the two processors to transfer data and instructions. Two types of mechanisms were considered: shared memory and FIFO buffers (First In First Out). Message passing with shared memory involves the use of mailboxes. FIFO buffers allow the message simply to be written into the buffer. The final implementation decided upon was the FIFO buffer mechanism because there is less processing per message and it is easier to implement.

4.1 Communication Requirements

This section describes the requirements for a Processor Communication Channel (PCC). Briefly, these requirements are:

- Message Passing.
- Mutual Exclusion.
- Speed.

Message passing is desirable because it allows the communication channel to be consistent with the Harmony operating system, which supports message passing. This allows a uniform presentation to the user, all communication takes place via messages. The requirement is for message passing at the User interface level. That is, the User only sends and receives messages.

The PCC must guarantee mutual exclusion at the Data Transfer Level where the two processors interact. Mutual exclusion is necessary because the two processors are sharing data between themselves. If mutual exclusion isn't enforced, then data could be written over or read before it is completely transferred.

The PCC should be the fastest possible while meeting the requirements as well as the additional constraints of cost, board space, and ease of implementation. With regard to speed, it is essential to use interrupts for signaling rather than busy waiting.

4.2 User Interface

All of the messages sent will be the same format which contains three fields:

- Message Type - allows differentiation of messages.
- Message Length - the number of words to follow.
- Data Field - contains the message itself.

The message type field is placed first to allow easier branching to subroutines on the signal processor. If the message length were first, it would have to be passed to the subroutines. There is no difference for user tasks because the message is contained in a high level structure. The message type and message length fields

are one word in length. They don't need to be one word in length (who needs 65,000 different message types?) but are one word to allow easier manipulation for 16-bit processors. If the message length and message type were one byte long so that they both fit into the same word, then they both would be read at the same time and additional data manipulations would be needed to separate the two values. It is much faster and easier to simply do two reads of one word each rather than one read of a single word and several extra manipulations to obtain the separate values. There are several message types that are pre-defined for use in all applications:

- Download - used to denote a message that contains programs and must be placed in program memory.
- Begin_Execution - commence execution of the program in program memory.
- Terminate_Execution - stop execution of the current program and enter a known state.

In addition, for the speech recognition application two other message types have been defined:

- Learn - enter the next word in the vocabulary.
- Recognize - perform speech recognition and return the best estimate of the spoken input.

4.3 Data Transfer

There will be separate processes to handle sending and receiving, running on each processor. Separate send and receive processes allow a simpler interface

to the processor communication channel, with one task overseeing half of the communication. Since interrupts are going to be used for signaling, making the processes tasks on the microprocessor allows them to wait for interrupts and still allow the main processes to continue execution. A more complete discussion is provided in chapter 5. These two tasks can be completely independent. In the case of the 68020, these processes will be separate tasks: SEND320 and RCVE320. User tasks wanting to send a message to the signal processor call the server task, which blocks the user. The server task passes the message to the SEND320 task, which writes the message into the processor communication channel. Once the message is written, the server task releases the user task.

An interrupt is generated when a message is received. RCVE320 waits for the interrupts and receives the message. Users wanting to receive a message, call the server task where they are blocked until a message arrives for them. Upon receiving a message, RCVE320 passes it to the server task and then waits for another. The server task will reply to the user task and pass along the message. It is most likely that transport tasks will be used to wait for messages because presumably user tasks won't want to be blocked waiting. If a user task is not waiting for a message which arrives, the message will be queued until that task calls to pick it up. When the user task calls, it will be passed the message right away and will not block.

On the 32020 side, the processes consist of SEND68, a subroutine, and RCVE68, an interrupt service routine (ISR). A process wishing to send a message to the 68020 calls the SEND68 subroutine and passes it the beginning address of the message to be sent. The SEND68 routine writes the message into the FIFO buffer,

returns and the process continues. When a message is to be received by the 32020, an interrupt is generated. RCVE68, an ISR on the 32020 is activated to receive the message. The user sets up a jump table to point to routines to deal with the expected messages. Based upon the message type, the ISR then calls the particular routine to deal with the message. This approach yields maximum flexibility and allows tailoring of the ISR for different uses. The user program can proceed to do what it must with the message: handle it, place a job on a queue, send a return message, or whatever. The user program can be long or short depending upon the demands of that particular environment.

An example of this is found in the speech recognition application. The speech recognition software operates in one of two states: Learn Mode or Recognize Mode. When a Learn message arrives, the ISR calls the Learn routine which reads the rest of the message, the ASCII values of the word to be learned, and proceeds to learn the word. When a recognize message arrives, the Recognize routine is called and words are recognized. There is no data in a Recognize message so the Recognize routine need not call the subroutine to obtain the message data.

4.4 Mailboxes Via Shared Memory

Since the two processors are located on tightly connected boards, they could share memory to transfer data. Single word transfers would take too much processor interaction since the processors must interact for every word transferred rather than every message and so they are more dependent upon each other. Two mechanisms, mailboxes implemented with shared memory and FIFO buffers, were considered to implement the PCC.

Message passing with shared memory involves the use of mailboxes to place and retrieve messages. The entire shared memory could be thought of as a single mailbox. Each processor would pass outgoing messages into the mailbox and pick up incoming messages.

If the incoming and outgoing data paths are separated then it is easier to have separate Send and Receive message processes running on each processor. This way each process will have a separate data area. In order to separate the data paths, the shared memory could be split in two mailboxes: one for messages traveling in one direction and the other for the opposite direction.

The idea of mailboxes can be expanded to include a Receive mailbox for every process on either of the two processors. However, the process running on the 68020 is a server task and it will be providing service to user tasks running elsewhere in the Harmony system as well as generating its own messages. These user tasks are not known in advance and could change over the life of the server. Therefore it is easier to have a single task or process running on the 68020 to handle incoming and outgoing messages. The N-mailbox idea can be generalized to allow one Receive mailbox per process running on the 32020 plus one mailbox for the message handler task running on the 68020.

The major advantage of the N-mailbox method is that it is possible to differentiate between messages for different processes. However, if there are many processes running on the 32020, then the number of mailboxes needed could be large. If the number needed is too large, the software overhead required to manage the mailboxes could be prohibitive and the space allocated to each mailbox would be small, perhaps not even enough to store all the messages destined for a particu-

lar mailbox. For example, the ESM application [9] has four independent channels and four mailboxes would be needed to differentiate the messages for each channel. If each mailbox needs to store three messages, each of which is 40 words, then 480 words of storage are needed for four channels. At 120 words per mailbox, even ten or twenty mailboxes are easily managed. However, if processes are dynamically created and destroyed, mailboxes for them must also be dynamically created and destroyed, which increases the software overhead and introduces the possibility of messages arriving for mailboxes which have been destroyed or not created. Thus the disadvantages outnumber the advantages and the two mailbox implementation would be chosen over an N-mailbox implementation.

Mutual exclusion between the two processors is necessary because the two processors are sharing data between them. Mutual exclusion is not guaranteed with shared memory. A Test-And-Set (TAS) instruction [6] would provide an efficient way of enforcing mutual exclusion but there isn't a TAS instruction available on the signal processor [23]. Therefore, it would be necessary to use Dekker's algorithm [6] to guarantee mutual exclusion. This is an algorithm which utilizes shared variables to make sure that two processes don't access the same data at the same time. It also provides fair, random access. Dekker's algorithm is implemented in software and provides mutual exclusion between processes running on the same or different processors. Since the two processes are running on different processors, additional measures must be taken to ensure data integrity at the hardware level. Data buffers and address latches must be included in the hardware design to allow both processors independent access to the shared variables. A logic circuit is needed to provide arbitration so that both processors don't access the shared

variables at the same time.

The time to access the shared memory is not negligible. First the process must execute Dekker's algorithm which consists of approximately 20 instructions for the signal processor. If a message of 10 words is to be sent or received, Dekker's algorithm would double the total execution time. The processor may then have to wait while the opposite processor finishes its access to the shared memory. This delay won't be longer than the time to read or write a single message. Once this is done, then the access to the shared data may occur.

Once the entire message has been written into the shared memory, an interrupt can be generated by writing into a status register. The generated interrupt signals to the opposite processor that a message has arrived. A message count is needed to keep track of the number of messages received and the number read. If another message arrived before the last one has been read out, the same interrupt would be generated, and so the receiving processor wouldn't know that another message has arrived, and the latest message would be lost. A message count will prevent this from happening. After a message has been sent, the counter is incremented; when it has been read, the counter is decremented. If the count is greater than zero then another interrupt is generated. This is easily implemented in shared memory by having a counter location.

4.5 FIFO Buffers

First In First Out (FIFO) buffers, implemented in hardware, solve the problem of mutual exclusion and contention through their design. Mutual exclusion is guaranteed by the hardware since one processor can write data into the FIFO

and the other can read data out at the same time. A separate FIFO is used for the opposite direction. Contention is no longer a problem because the reading and writing can occur simultaneously even on the same FIFO buffer. By having both processors reading and writing at the same time, the message rate could be doubled. The FIFO buffers utilized are new devices and can store 512 words. This is large enough to allow several messages to be buffered.

The FIFO buffers are fast enough to meet the access time of the 32020 and so will allow one cycle access to read or write data. The buffers are fast enough to allow no-wait-state execution with the 68020 as well. Recall that to access shared memory, first Dekker's algorithm must be executed and then, possibly, the processor must wait while the other processor finishes its access to the shared memory. Thus FIFO buffers have a shorter access time than shared memory.

The processors are notified of arriving messages by an interrupt, which could be generated by the FIFO buffer "empty" flag. The "empty" flag changes state when the first word of the message has been written into the buffer. However, the 32020 could possibly read data out of the buffer faster than the 68020 could put it in since the 32020 has a faster cycle time than the 68020. Garbage data would be the result. Therefore, a status register is needed to generate an interrupt. The status register would be written into after the entire message had been written into the buffer. As with shared memory, it is necessary to have a memory count. Up/down counters are used because no shared memory is available. One counter is used for each message transfer direction and they are accessible to both processors.

4.6 Final Implementation

Both the shared memory and FIFO buffer implementation met the requirements for the processor communication channel. The final implementation is the FIFO buffer implementation. They will be used as described previously with additional protection to prevent the buffers from overflowing. The decision to use FIFO buffers was based on four factors, two in favor and two against:

- **Speed** – The FIFO implementation has no software overhead. The software overhead for the shared memory implementation includes the execution of Dekker's algorithm and the organization of the message queue.
- **Ease of Implementation** – In the case of FIFO buffers, only hardware is needed. It is straightforward to implement using memory-mapped buffers. The shared memory implementation is more involved because of the need for both software and hardware. The software for Dekker's algorithm and the data management must be written. Data and address buffers are needed for both processors as well as a hardware arbitration scheme to prevent simultaneous memory accesses.
- **Board Space** – It has been calculated that the FIFO implementation will take approximately 25% more area than the corresponding shared memory implementation.
- **Cost** – The FIFO implementation is approximately five times more expensive than the shared memory implementation.

It is felt that the advantages of speed and ease of implementation outweigh the disadvantages of cost and the increase in board area required.

To prevent the FIFO buffers from overflowing, the additional constraints of a maximum message length and a maximum number of outstanding messages have been introduced. If the maximum message size times the maximum number of messages outstanding is less than the total message space; then the message space will never become full. If a processor sends the maximum number of messages, then it can't send any more messages until the opposite processor has read out at least one message. How does the processor know when it can send again? The processor could check periodically to see if a message has been read out but that's busy waiting. A better solution is to have an interrupt generated when more room is available. In the final design, the "OK_TO_SEND" interrupt is generated by a logic circuit with access to the message count for the FIFO buffer. The sending processor receives the interrupt and can then send its message. If the need to send a message has passed, then the "OK_TO_SEND" interrupt can be ignored.

Chapter 5

Server Task

The basic requirement of the Signal Processing Server (SPS) is to provide an interface for the signal processing subsystem to the rest of the system, which uses the Harmony operating system. This interface must provide communication between Harmony tasks and the signal processor via the Processor Communication Channel (PCC) and allow Harmony tasks to exercise control over program execution on the signal processor. This chapter describes the server task design and functionality.

Communication between the signal processor and Harmony tasks is via the Processor Communication Channel (PCC). As described in the PCC requirements, the SPS will perform the actual reading and writing of the PCC messages, and take care of the PCC interrupts. A consistent interface is provided for user tasks because the server task has message-based communication and utilizes blocking sends and receives.

The SPS will allow a Harmony task to exercise control over program execution on the signal processor. Program control includes downloading code, starting execution, terminating execution, and changing the program which is executing on the signal processor.

In addition to the general server requirements, speech processing places additional constraints on the system. Speech processing is characterized by high I/O data rates and lower data rates across the server interface. These data transfers must occur in real-time. In the case of speech recognition, the recognized words pass across the server interface, each of which is contained in an eight word message. During speech synthesis all of the LPC coefficients will be passed across the server interface, one LPC coefficient every 125 microseconds. Still, these are not high data rates and can easily be managed by the processor communication channel.

5.1 Server Design

As well as providing a user-friendly interface to the signal processor, the server task must be fast and efficient. By being user friendly, the server should be simple to understand, generalized for most applications, and reusable. The server will be an aid when prototyping new systems because it will provide the structure and modularity (by way of existing tasks) to new systems. The programmer can reuse all of the server software and only has to write the user or application task. Speed and efficiency are necessary to provide the greatest throughput across the server interface. By having as few sends and receives as possible, by limiting the number of tasks, and by optimizing the system, this goal can be achieved.

Unfortunately, the two aims of user friendliness and speed and efficiency are somewhat opposed to each other. If the server is generalized and reusable, it will have some unnecessary overhead within it. However, the programming effort required to get a new application up and running is minimized. An optimized

application takes more effort on the part of the programmer but the reward is greater efficiency. Taking this into account, the server was configured in several different ways. These are

- Single-User Prototype System.
- Optimized Single-User System.
- Optimized Multi-User System.

It is expected that most of the systems developed through the ARTT project will first undergo prototyping and later be optimized. This is why the prototype system is included.

The single-user prototype system is shown in figure 5.1. The SERVER task design is based on the administrator concept [7]. The USER tasks call the administrator, who passes the request on to the worker tasks. In this case, the workers are the SEND320 and RCVE320 tasks which transfer data to and from the FIFO buffers. The workers send to the administrator, who replies with the request to be completed. In this way the administrator only receives messages, and consequently does not block. The administrator also knows which workers are free, which are busy, and can queue requests until the worker is free. This allows the administrator to handle multiple requests. The SEND320 task accepts messages from the SERVER task and writes them into the FIFO buffer. If the buffer becomes full, the SEND320 task waits for the "OK_TO_SEND" interrupt. The RCVE320 task waits for the "MESSAGE_ARRIVED" interrupt, reads the message from the FIFO buffer, and passes it to the SERVER task. The interrupt handler is an assembly language routine which actually receives the processor communication

Figure 5.1: Single-User Prototype System.

channel interrupt and decides which task to call. A transport task can be included so that the USER task doesn't block when calling the SERVER task. It doesn't have to be used if the USER task can block without consequence. A transport task is not used in the the speech recognition application.

The advantage of this design is that it is general and completely reusable. To set up a new system, a programmer need only write the USER task, the application program for the signal processor, and define the message types for communication between the USER task and the signal processing application program. The disadvantage is the overhead incurred in message passing: each message is copied three times (four times if the transport task is used) and the total time needed to write a message into the FIFO buffer is 10.4 milliseconds. This compares with 2.0 milliseconds for a single send, receive, reply cycle.

The single-user prototype system could be expanded to accommodate more than one signal processing subsystem, all accessed via the same SERVER task as in figure 5.2. If each subsystem has its own processor communication channel, then only a SEND320 and RCVE320 task pair is needed for each additional signal processor. If more USER tasks wished to use the SPS, they could all use the same send queue for a particular signal processing subsystem because all the messages are going to the same signal processor. The distribution of received messages is more difficult. One transport task could be used to accept returning messages and deliver them to the appropriate USER tasks. However, if the transport task became blocked at a particular USER, then none of the other USER tasks would receive any messages. It is better if each USER task picks up its own messages and so one receive queue is needed per USER task. If the users can't afford to be

Figure 5.2: Multiple Signal Processing Subsystems.

blocked then they can each use their own transport task to pick up their messages.

The optimized single-user system, shown in figure 5.3, is more efficient than the prototype system, but it is also more specialized. This design is more specialized because both the Server task and Send320 task functionality have been moved into the USER task as procedures. It is more efficient than the prototype system because the message copying has been reduced from three or four times per message to only once because the USER task will write its messages into the FIFO buffer and read them out itself. It is more efficient because the number of tasks has been reduced and so there will be fewer task context switches (two as opposed to eight or ten). A variant of this design can be used if there is a high probability of the FIFO buffer filling up and if the USER task should not be blocked waiting for the "OK_TO_SEND" interrupt. The SEND320 task can be used to wait for this interrupt and notify the USER task when it occurs. But, the USER task will still write the messages into the FIFO buffer to save message copying. Another variant can be used if there is no parallelism and the USER task knows when to expect messages. In this case, the RCVE320 task isn't needed. But if the RCVE320 task is included, it can wait for the "MESSAGE_ARRIVED" interrupt and free the USER task to perform other functions. This design is very easily expanded to accommodate more signal processing subsystems controlled by a single USER task which includes the SERVER functionality. The Send procedure needs to know which subsystem to send to. The interrupt handler can decide and tell the RCVE320 task which subsystem the message has been received from.

If more than one USER task is to use one signal processing subsystem, then the SERVER task functionality cannot be included in one of the USER tasks, and

Figure 5.3: Optimized Single User System.

a separate SERVER task is essential. The optimized multi-user system is shown in figure 5.4. The SERVER task could be used as a semaphore with the USER tasks calling to acquire and release the FIFO buffer to send messages. However, calling and releasing a semaphore task will take four context switches and one message copy while simply passing the message to the SERVER task takes only two context switches and two message copies. A context switch involves saving 35 words onto the stack. If the messages are less than 70 words in length, there is an advantage in allowing the SERVER task to write the messages into the FIFO buffer. The typical message size is 40 words for the ESM application and so the SERVER is used to write the messages into the FIFO buffer. A RCVE320 task is needed to wait for the "MESSAGE_ARRIVED" interrupt and to pick up the PCC messages because it is more efficient than the USER tasks picking up their own messages for the same reasons as sending messages. A transport task is needed to transfer messages between the Server and RCVE320 tasks because neither of them can risk sending and blocking. As with the optimized single user system, the optimized multi-user system is expandable to accommodate more signal processing subsystems by using the same Send procedure and RCVE320 task.

Server Testing

The single-user prototype system was tested using a Harmony simulator which runs on a VAX computer or Sun workstation. The system performed as expected and successfully passed messages to, accepted interrupts, and received messages from a simulated signal processing task. The SPS couldn't be tested on the hardware built because the general-purpose microprocessor didn't function properly and consequently the Harmony operating system couldn't be ported to run on it. How-

Figure 5.4: Optimized Multi-User System.

ever, small assembly language routines could be executed and all of the low-level routines for the PCC were exercised and found to work as expected. Once the Harmony operating system is ported to the general-purpose microprocessor, the SPS should perform its functions as expected.

Chapter 6

Application to Speech Processing

In this chapter, a brief summary of the state of speech recognition today is presented, the speech recognition algorithms are discussed, and the implementation and testing of these algorithms is described.

6.1 Speech Recognition Today

This section summarizes the current state of speech recognition today. Speech recognition can be broken down two ways:

- Speaker-dependent versus speaker-independent.
- Independent-word versus continuous speech recognition.

Speaker dependency refers to whether the system is trained or not. If the speech recognition system must be trained by a particular speaker then the system is speaker-dependent. Training provides more accurate recognition capabilities because more precise word templates can be created. A speaker-independent system ideally would recognize a particular word no matter whom it was spoken by.

An independent-word speech recognition system can only recognize a single word at a time and forces the user to speak with distinct pauses between the words.

Conversely, a continuous recognition system allows the user to speak normally, without pauses. An independent-word speech recognition system is simpler to implement since the word boundaries are easier to detect because of the pauses, and the recognition rate doesn't have to be quite as fast because the speaker isn't speaking as quickly. In this project, a speaker-dependent, independent-word speech recognition system was implemented.

A survey of speech recognition systems [17] reviews the capabilities of current systems. The vocabulary size for practical application of an independent-word, speaker-independent system is approximately 10 - 50 words with an average recognition accuracy of 95%. The vocabulary size associated with a practical independent-word, speaker-dependent system is approximately 100 - 200 words with an average recognition accuracy of 98.5%. Current continuous speech recognition systems have recognition accuracies which range from 44% to 95% and widely varying sizes of vocabularies. Continuous speech recognition is still in the experimental stage and many problems need to be solved before it becomes realizable.

There are several applications for speech recognition systems [5] such as:

- Voice Data Entry - can be used rather than entering data by typing it at a terminal. This will ultimately result in a voice-activated typewriter.
- Voice Command - can control appliances and machines without being near them. It would be easier to adjust a television set by speaking than with the current remote control units and there is no control box to misplace. Listening light switches are also possible. Machines could be controlled or at least deactivated just by shouting "STOP".

- Speaker Identification - a person can be identified quickly and accurately, no need for security badges anymore. The system can provide voice-controlled access to secure areas.
- Conversational Applications - much development needs to be done on the natural language processing side of things but applications abound: telephone operators, reservation systems, teachers, and, possibly, friends.

6.2 Speech Recognition Algorithms

The speech recognition system, built to demonstrate the server design, takes spoken words as input and gives the best estimate of the input word with respect to the stored words in its vocabulary. A block diagram of the system is shown in figure 6.1 [16] and is characterized by three main blocks:

- Feature Extraction.
- Pattern Similarity Detection.
- Decision.

Feature Extraction is a data reduction technique in which the features of a speech signal represented by a large number of samples can be replaced by a much smaller feature set which performs the same task. The Pattern Similarity Detection block performs the actual comparison between the test pattern and the reference patterns. The test pattern consists of the feature vectors of the unknown spoken input and the reference patterns are previously stored words which make up the speech recognizer's vocabulary. Because speaking rates vary greatly, time alignment must be performed as well as distance computation. These two functions

Figure 6.1: Speech Recognition System.

may be performed at the same time using the dynamic time warping method [18], the result of which is a time-normalized distance between the test pattern and a reference pattern. The basic unit of comparison in the speech recognition system is the frame. Each word is composed of a series of frames, and words are compared by summing the time-normalized distances between corresponding frames of the words. Dynamic time warping is performed on the test pattern and each of the reference patterns in turn. Then the Decision Block compares all of the time-normalized distances calculated and chooses the smallest. The reference pattern with the smallest distance is selected as the best estimate of the spoken input.

The speech recognizer operates in two states:

- Learn Mode.
- Recognition Mode.

Learn mode is used to build up a vocabulary. A word is spoken, feature extraction is performed, and the resulting feature vector is stored in the system's vocabulary. There is no output from this state. The actual speech recognition occurs in recognition mode. Feature extraction is performed on an unknown spoken input, giving a feature vector called the test vector. This test vector is compared to all of the reference vectors during pattern similarity detection, and the reference pattern with the smallest distance is output from the decision block. This is the best estimate of the unknown spoken input with respect to the system's vocabulary.

6.2.1 Feature Extraction

The process of turning the large number of data samples into a much smaller feature set which still retains all of the signal properties is called feature extraction.

The speech recognition system described in this chapter is based on a feature set which uses linear predictive coding (LPC) coefficients [16]. Other methods of extracting features [5] are zero-crossing rates and formant analysis. The LPC model was selected because it is frequently used and the algorithms were readily available. Feature extraction processing is composed of

- Input.
- Endpoint Detection.
- Preprocessing.
- Autocorrelation Analysis.
- LPC Analysis.

Input

The user speaks into a microphone. The resulting signal is amplified, filtered by a standard codec filter, and sampled at 8 kHz. The rest of the analysis is performed using these digital samples.

Endpoint Detection

Endpoint detection is done to detect the beginnings and endings of words which eliminates unnecessary processing. Endpoint detection is based on the combined energy of three frames of samples compared to a threshold energy level.

$$Energy = \frac{1}{300} \sum_{i=0}^2 \sum_{j=1}^{100} (X_{100i+j})^2 \quad (6.1)$$

The frames are 300 samples long which corresponds to a time duration of 37.5 milliseconds. The frames are spaced 100 samples apart which means that adjacent frames overlap by 200 samples and each sample contributes to three consecutive frames. Frame overlap allows for smoothing between feature vectors. When the energy level rises above the threshold value, the word has started and all of the following frames undergo feature extraction. Once the energy level falls below the threshold again, the word has ended and feature extraction is completed.

Preprocessing

Preprocessing involves filtering the input signal with a preemphasis filter, sorting the signal into frames, and applying a Hamming window to each frame. The preemphasis filter is used to spectrally flatten the input signal. It has a transfer function

$$H(z) = 1 - az^{-1} \quad (6.2)$$

where typically $a \simeq 0.95$ [16] and so

$$\hat{X}(n) = X(n) - 0.95X(n-1) \quad (6.3)$$

The resulting signal $\hat{X}(n)$ is blocked into frames for feature measurement. Lastly each frame is windowed by a Hamming function given by

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \quad (6.4)$$

where the frame length, N , is 300 samples. Let L be the number of frames in a word and $X_l(n)$ be the n^{th} sample of the l^{th} frame of speech. The preprocessing computations can be summarized as follows

for $l = 0$ to $L - 1$

for $n = 0$ to 299

$$\hat{X}_l(n) = [X_l(100l + n) - 0.95X_l(100l + n - 1)] * w(n);$$

Autocorrelation Analysis

A P^{th} order autocorrelation analysis is performed on each frame of windowed data. P is typically between 8 and 12. In this case, P is chosen to be 8 so as to minimize computation and memory requirements [20]. For the l^{th} frame

$$R_l(p) = \sum_{n=0}^{N-1-p} \hat{X}_l(n) \hat{X}_l(n+p) \quad (6.5)$$

where $p = 0$ to P . Next the autocorrelation coefficients are normalized so that $R_l(0) = 1$, as follows

$$R'_l(p) = \frac{R_l(p)}{R_l(0)} \quad (6.6)$$

Normalization is required to allow the later computations in the LPC analysis to use the full integer precision of the processor. Normalization also minimizes the effects of different speaking levels. The autocorrelation computations can be summarized as follows

for $p = 0$ to P

begin

$$R[p] = 0;$$

for $n = 0$ to $N - p - 1$

$$R[p] = X[n] * X[n+p] + R[p];$$

end;

LPC Analysis

Linear predictive coding assumes that a speech sample can be approximated as a linear combination of previous speech samples. The predictor coefficients can be determined by minimizing the squared difference between the actual speech samples and the linearly predicted ones. These predictor coefficients are used as the feature set to represent the speech signal. One set of predictor coefficients is calculated for each speech frame. Assume that the transfer function of a speech production model [19] can be represented as

$$H(z) = \frac{A}{1 - \sum_{k=1}^p a_k z^{-k}} \quad (6.7)$$

Then the speech samples, $x(n)$, are related to the excitation, $\delta(n)$, by

$$x(n) = \sum_{k=1}^p a_k x(n-k) + \delta(n) \quad (6.8)$$

If a linear predictor model is used to predict the speech samples, $\tilde{x}(n)$, then

$$\tilde{x}(n) = \sum_{k=1}^p \alpha_k x(n-k) \quad (6.9)$$

and the corresponding predictor error, $\epsilon(n)$, is

$$\epsilon(n) = x(n) - \tilde{x}(n) = \sum_{k=1}^p a_k x(n-k) + \delta(n) - \sum_{k=1}^p \alpha_k x(n-k) \quad (6.10)$$

Now, if $a_k = \alpha_k$, then $\epsilon(n) = \delta(n)$ and, between the excitation impulses of voiced speech, the prediction error will be very small.

To determine the predictor coefficients, the average squared prediction error over a small segment of speech is minimized

$$E = \sum_{n=a}^{a+N} (x(n) - \tilde{x}(n))^2 \quad (6.11)$$

where a is arbitrary and $x(a)$ to $x(a + N)$ is a small segment of speech in the vicinity of the sample.

$$E = \sum_{n=a}^{a+N} (x(n) - \sum_{k=1}^p \alpha_k x(n-k))^2 \quad (6.12)$$

The values of α_k that minimize E can be obtained by setting $\partial E / \partial \alpha_i = 0$, $i = 1, 2, \dots, p$, giving

$$\sum_{n=a}^{a+N} x(n-i)x(n) = \sum_{k=1}^P \alpha_k \sum_{n=a}^{a+N} x(n-i)x(n-k) \quad (6.13)$$

If

$$\varphi(i, k) = \sum_{n=a}^{a+N} x(n-i)x(n-k) \quad (6.14)$$

then

$$\sum_{k=1}^P \alpha_k \varphi(i, k) = \varphi(i, 0) \quad i = 1, 2, \dots, p \quad (6.15)$$

This set of p equations in p unknowns can be solved for the unknown predictor coefficients that minimize the average squared predictor error.

However, $x(n)$ can be determined in two ways which leads to two ways to solve for α_k . Basically, if the speech samples are not windowed the covariance method is used since the matrix $\varphi(i, k)$ is very similar to a covariance matrix. However, if the samples are windowed then the autocorrelation method is used. This is evident in

$$\varphi(i, k) = \sum_{n=a}^{a+N} x(n-i)x(n-k) = \sum_{n=a-i}^{a+N-i} x(n)x(n-k+i) = r(i-k) = r(k-i) \quad (6.16)$$

Thus

$$\sum_{k=1}^p \alpha_k r(|i-k|) = r(i) \quad i = 1, 2, \dots, p \quad (6.17)$$

This equation can be solved using the Levinson method [10] but, in this thesis, was solved by using the Leroux-Geugen (L-G) [12] matrix inversion to derive the

reflection coefficients which are then transformed into the predictor coefficients with the Step-Up algorithm [12]. The L-G algorithm is used to derive the reflection coefficients because all values calculated remain less than one; a very important property for a fixed point arithmetic implementation. That is, if all values remain less than one then there is no possibility of overflow. The L-G algorithm can be summarized as follows

```

 $k[1] = -R[2]/R[1];$ 
 $BIM1[1] = R[2];$ 
 $BIM1[2] = R[1] + k[1] * R[2];$ 
for  $i = 2$  to  $P$ 
begin
     $YI = R[i + 1];$ 
     $BI[1] = YI;$ 
    for  $j = 1$  to  $i - 1$ 
begin
         $BI[j + 1] = BIM1[j] + k[j] * YI;$ 
         $YI = YI + k[j] * BIM1[j];$ 
         $BIM1[j] = BI[j];$ 
    end;
     $k[i] = -YI/BIM1[i];$ 
     $BIM1[i + 1] = BIM1[i] + k[i] * YI;$ 
     $BIM1[i] = BI[i];$ 
end;

```

The Step-Up algorithm can be summarized as follows

```

a[1] = 1;
a[2] = k[1];
for i = 2 to P
begin
    for j = 1 to i
    begin
        JB = i - j + 1;
        B[j] = a[JB];
    end;
    for k = 2 to i
    begin
        a[k] = a[k] + k[i] * B[k - 1];
    end;
    a[i + 1] = k[i];
end;

```

6.2.2 Pattern Similarity Detection

Pattern similarity detection is performed once feature extraction has been completed. A comparison must be made between the test pattern T , which represents the spoken input, and a reference pattern R , which represents a pattern from the vocabulary. The two patterns can be represented as

$$T = (T_1, T_2, \dots, T_{N_t}) \quad (6.18)$$

and

$$R = (R_1, R_2, \dots, R_{N_r}) \quad (6.19)$$

where N_t and N_r are the number of frames in the test and reference patterns (words) respectively. T_i and R_i are the i^{th} feature vector in each pattern. Because speaking rates vary, it is unlikely that N_t will equal N_r and so some sort of time alignment is necessary. This is usually performed at the same time as the distance computation using a method known as dynamic time warping. Earlier attempts at eliminating time differences used linear transformations of the time axes [18], but these were insufficient for dealing with the speech signals.

Dynamic Time Warping

Dynamic time warping is a method in which a non-linear warping function is applied to both test and reference patterns in an attempt to perform time alignment and distance computation. The warping function maps the time axes of the test and reference patterns onto a normalized time axis. If there was no timing difference between the two patterns then the warping function would be a diagonal line, $i = j$. However, the greater the timing difference the more the function deviates from the diagonal line. This is illustrated in figure 6.2.

The objective of dynamic time warping is to determine the time-normalized distance between the test and reference patterns, $D(T, R)$. A simple euclidian distance can be used to measure the distance between any two feature vectors. For two vectors T_i and R_i , the euclidian distance, $d(T_i, R_i)$, is given by

$$d(T_i, R_i) = \sum_{p=1}^{P-1} (T_i(p) - R_i(p))^2 \quad (6.20)$$

where there are $P - 1$ elements in each feature vector.

Figure 6.2: Dynamic Time Warping Function

The measure of goodness of the warping function, $w(k)$, is the summation of distances from the diagonal line. Thus the time-normalized distance between the test and reference patterns can be described as

$$D(T, R) = \min_{w(k)} \frac{\sum_{k=1}^K d(T(k), R(k))w(k)}{\sum_{k=1}^K w(k)} \quad (6.21)$$

where the denominator is used to compensate for the number of points on the warping function. The above equation can be solved using dynamic programming methods. The dynamic programming equation for $D(n, m)$ must be calculated in ascending order with respect to the coordinates n and m which represent the frames of the test and reference patterns respectively. The dynamic programming algorithm can be summarized as follows

for $m = 1$ to Nt do

for $n = m - r$ to $m + r$ do

if $(n > 0)$ and $(n < Nr)$ then

$$D(n, m) = \min \begin{cases} D(n-1, m-2) + 2d(n, m-1) + d(n, m) \\ D(n-1, m-1) + 2d(n, m) \\ D(n-2, m-1) + 2d(n-1, m) + d(n, m) \end{cases}$$

This will yield the result $D(Nt, Nr)$ which is the time-normalized distance between the test and reference patterns. The parameter r is called the window length and is used to constrain the warping function from becoming too steep or too shallow. If this wasn't the case, then a short test pattern could be matched with a much longer reference pattern or vice versa. The window length used in the calculations was 6 frames [18].

6.2.3 Decision

The decision is simply a comparison of all the resulting distances, one for each possible pattern. The smallest is selected and the word corresponding to that reference pattern is passed to the calling process.

6.3 Implementation and Testing

The speech recognition algorithms presented were initially implemented in a high level language using floating point arithmetic on a VAX computer to be certain that the algorithms performed as expected without any constraints imposed by fixed-point arithmetic, real-time operation, or the TMS32020 instruction set. They were tested using segments of speech data obtained from another student. The speech recognizer performed as expected, identifying the speech segment correctly each time. Next the speech recognition program was changed to use fixed point arithmetic. Scaling problems were encountered but resolved.

The speech recognition algorithms were then written in TMS32020 assembly code so they could be used on the signal processor. The algorithms were tested using both a simulator and an in-circuit emulator. However, significant overflow problems were encountered in the LPC routine, which used the Levinson algorithm, and couldn't be resolved. The LG and Step-Up algorithms were used instead. The software for these two algorithms obtained from Professor Morris was for the TMS32010 and once upgraded to the 32020 worked fine. Finally, the speech recognition algorithms were tested in real-time using live speech. A vocabulary of five words, consisting of the numbers one through five, was created and the spoken input was correctly identified. However, if a word not in the vocab-

ulary was spoken, an error result was returned as expected. Unfortunately, the speech recognition system has not been thoroughly tested because the hardware upon which the SPS executes [9] is not operational. The vocabulary of five words and associated testing was done with the use of an in-circuit emulator. Assembly language routines had to be executed to perform the functions of learning and recognizing words. It is expected that the actual vocabulary will be less than that predicted in Chapter 2 because of processing time taken up by data manipulations not associated with computation of the algorithms themselves.

Algorithm	Estimated	Actual
Endpoint Detection	0.05	0.03
Preprocessing	0.84	0.92
Autocorrelation Analysis	3.24	3.00
LPC Analysis	0.20	0.52
Feature Detection	4.33	4.47

Figure 6.3: Comparison of Execution Times per Frame in msec.

Figure 6.3 provides a comparison of the estimated and actual execution times for the Feature Detection algorithms. The actual execution time for Feature Detection as a whole is greater than the estimated time because of the time spent moving data in memory. Examples of data movement are moving the data from the input buffer and performing preprocessing, and blocking the data into frames.

The time spent performing autocorrelation is less because one of the coefficients, $R(0)$, could be calculated by pipelining the TMS32020 instruction SQRA [23], rather than having a loop for all the instructions. There are 300 samples per

frame and eight coefficients to be determined. Ideally, the autocorrelation coefficients could be determined in a little more than 2700 clock cycles (0.54 msec). The TMS32020 has an instruction, MAC [23], which can multiply and accumulate, and update both the multiplier and multiplicand pointers in a single cycle when pipelined. But, the MAC instruction can only be used with the TMS32020's internal program memory which is only 288 words in size. Thus the autocorrelation routine was implemented using external memory and a loop because a pointer to the multiplier has to be updated, a multiply performed, which updates the pointer to the multiplicand. So three instructions are needed for each multiply (the third is the branch instruction) but since the data is in external memory each instruction takes two cycles. Therefore, 14,700 cycles (2.94 msec) are needed to calculate all the coefficients. This is very close to the 15,018 cycles (3.00 msec) actually needed. The autocorrelation routine could perhaps be optimized by moving data points into and out of the internal program memory since a six-fold speed increase is obtained with the MAC instruction.

Chapter 7

Conclusion

The aims of this thesis were to provide digital signal processing capabilities to a multiprocessor and provide a simple interface to these capabilities. By comparing the estimated execution times of the speech recognition algorithms for a general-purpose microprocessor and a digital signal processor, it was shown that a digital signal processor is needed to perform speech recognition with a usable-sized vocabulary. A design for the digital signal processing subsystem was discussed and a final design decided upon. This design is comprised of a general-purpose microprocessor to execute the Harmony operating system and interact with the rest of the multiprocessor system, a digital signal processor to provide the digital signal processing capabilities, and an analog I/O block to interface with the outside world by obtaining signal samples and distributing them. The applicability of the server concept to provide a simple interface was shown and a signal processing server (SPS) was implemented to run on the general-purpose microprocessor. A processor communication channel (PCC) was defined to allow the passage of data and commands between the general-purpose microprocessor and the digital signal processor. It interacts with both the SPS and the interrupt service routines on the digital signal processor. Finally, speech recognition algorithms were imple-

mented and executed to demonstrate the digital signal processing capabilities of the subsystem.

- The work can be continued by optimizing the speech recognition algorithms further to provide a greater vocabulary, and by implementing other signal processing applications, such as a modem or real-time FFT routines, using the signal processing subsystem.

Bibliography

- [1] J. Baer, *Computer Systems Architecture*, Computer Science Press, 1980.
- [2] A. Birrell and B. Nelson, "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, pp. 39-59, Feb. 1984.
- [3] R.J.A. Buhr, *System Design with Ada*, Prentice-Hall, 1984.
- [4] P. Cashin, "Inter-process Communication", *BNR Report*, 1980.
- [5] J. P. Cater, *Electronically Hearing: Computer Speech Recognition*, Howard W. Sams and Co. Ltd., 1984.
- [6] H. M. Dietel, *An Introduction to Operating Systems*, Addison-Wesley Publishing Co., 1984.
- [7] W. M. Gentleman, "Message Passing Between Sequential Processes: the Reply Primitive and the Administrator Concept", *Software - Practice and Experience*, volume 11, pp. 435-466, 1981.
- [8] W. M. Gentleman, *Using the Harmony Operating System*, 1984.
- [9] L. Hickman, *Design of a Parallel Signal Processing Architecture for ESM*, M.Eng. Thesis, Carleton University, March 1987.

- [10] J. Makoul, "Tutorial on Linear Prediction", *Proceedings of the IEEE*, volume 63, number 4, April 1975.
- [11] J. Mick and J. Brick, *Bit-slice Microprocessor Design*, McGraw-Hill, 1980.
- [12] L.R. Morris, J. Bosloy, and P. Barszczewski, "Algorithm Selection and Software Time/Space Optimization for a DSP Micro-Based Speech Processor for a Multi-Electrode Cochlear Implant", *Proceedings of ICASSP*, 1987.
- [13] L. R. Morris, "Digital Signal Processing Microprocessors: Forward to the Past", *IEEE Micro*, Dec. 1986.
- [14] A. Oppenheim and R. Schaffer, *Digital Signal Processing*, Prentice-Hall, 1975.
- [15] L. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, 1975.
- [16] L. R. Rabiner and S. E. Levinson, "Isolated and Connected Word Recognition - Theory and Selected Applications", *IEEE Transactions on Communications*, volume COM-29, pp. 621-659, May 1981.
- [17] S. Saito and K. Nakata, *Fundamentals of Speech Signal Processing*, Academic Press, 1985.
- [18] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, volume ASSP-26, pp. 43-49, Feb. 1978.
- [19] R. Schaffer and L. Rabiner, "Digital Representation of Speech Signals", *Proceedings of the IEEE*, volume 63, number 4, April 1975.

- [20] K. Siomalas, *A Case Study of Speech Recognition Algorithms*, ARTT 85-8, ARTT Project, Department of Systems and Computer Engineering, Carleton University.
- [21] *Motorola MC68010 Virtual Memory Processor*, Motorola Incorporated, 1983.
- [22] *Motorola MC68020 32-bit Microprocessor User's Manual*, Second Edition, Motorola Incorporated, 1985.
- [23] *Texas Instruments TMS32020 User's Guide*, Texas Instruments Incorporated, 1985.
- [24] *Digital Signal Processing Applications with the TMS320 Family*, Texas Instruments Incorporated, 1986.

END

22|06|88

FIN