Abstract

Title of Thesis: Data Acquisition Interface of a VLSI Cochlea Model

Name of degree candidate: Thomas G. Edwards

Degree and year: Master of Science, 1993

Thesis directed by: Associate Professor Shihab Shamma
Department of Electrical Engineering

Computer models of cochlear processing take exceedingly long times to run, even for short data sets. A data acquisition system was developed for a new switched-capacitor VLSI cochlea model chip, in order to rapidly perform cochleal processing on digitzed speech samples. The system is capable of processing very long speech samples. Processing is in near-real-time, though it takes about 2 minutes per second of speech to write the large amount of data to a hard drive. Software has also been developed to convert the output data into a form readable by the ESPS digital signal processing package from Entropic Speech, Inc.

Data Acquisition Interface of a VLSI Cochlea Model

by

Thomas G. Edwards

Thesis submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1993

c.1
MD
Electrical Engineering Dept.

Advisory Committee:

Associate Professor Shihab Shamma, Chairman/Advisor
Professor Martin Peckerar
Assistant Professor Linda Milor

Maryland
LD
3231
.M70m
Edwards,
T.G.

© Copyright by

Thomas G. Edwards

1993

Table of Contents

Section	Page
List of Tables	**iv**
List of Figures	**v**
1 Introduction	**1**
2 The Cochlea Model Chip	**2**
3 Design of the Interface System	**5**
3.1 Design Criteria	5
3.2 Hardware Design	6
3.2.1 Analog Interface Circuitry	6
3.2.2 A/D and D/A Interface	9
3.3 Filter Characterization Test Equipment	12
3.4 Data Acquisition Software	12
3.4.1 Software Design Criteria	12
3.4.2 WBFLASH-12 Description	14
3.4.3 Initialization	15

3.4.4 Recording Data 15
3.4.5 Previewing data 18
3.4.6 Processing data 18
3.4.7 Demo Mode 19
3.5 Network Data Transfer 20
3.6 Data Conversion Software 21

4 Results and Conclusions **24**
4.1 Frequency Response of the Cochlear Channels 24
4.2 Single Sinusoid Tests 24
4.3 Two Tone Test 26
4.4 Single vowel tests 26
4.5 Isolated Words 34
4.6 Long speech example 39
4.7 Discussion of Results 42
4.8 Future research 44
4.9 Conclusions 45

A Source Code for cochint.c **46**

List of Tables

Number Page

3.1 Specifications of WB-FLASH12 Card 11

List of Figures

Number		Page
3.1	Reference Clock Generators	8
3.2	Nonoverlapping Clock Generator Circuit	9
3.3	Data Acquisition System Block Diagram	13
3.4	cochint.c program flowgraph	16
3.5	Slice.c data reshuffling	23
4.1	Magnitude response for channels 34, 59, and 64	25
4.2	Response to single 200 Hz tone (Total time 48ms)	27
4.3	Response to single 800 Hz tone (Total time 48ms)	28
4.4	Response to single 1.5 kHz tone (Total time 24ms)	29
4.5	Response to single 2 kHz tone (Total time 24ms)	30
4.6	Response to single 3 kHz tone (Total time 12ms)	31
4.7	Response to 600 Hz tone added with 2 kHz tone (Total time 48ms)	32
4.8	Response to 1 kHz tone added with 3 kHz tone (Total time 48ms)	33
4.9	Response to the /iy/ vowel sound (Total time 48ms)	35
4.10	Response to the /oo/ vowel sound (Total time 48ms)	36
4.11	Response to the /A/ vowel sound (Total time 48ms)	37

4.12 Response to the spoken word "you" (Total time 480ms) 38

4.13 Response to the spoken word "free" (Total time 600ms) 40

4.14 Response to the spoken word "bay" (Total time 600ms) 41

4.15 Response to spoken words "four five six seven" (Total time 2.4s) . 43

Chapter 1

Introduction

It has been suggested that cochlear models might be appropriate for many types of speech processing [1]. Investigators have developed mathematical models of the cochlea which run on workstation-style computers, but unfortunately, they often take hours to process a single second of speech. In [2] special digital signal processing hardware was used to try to speed up cochlear processing, but this method did not significantly decrease the time required to process speech data. It was recognized that an analog VLSI approach to the problem would allow for near-real-time cochlear processing, and a VLSI model was developed by Lin [3]. But in order to be useful, a data-acquisition system had to be developed to record the data from the cochlea model, and convert it to a form useful by popular signal processing software. This thesis is concerned with the development and testing of this system.

Chapter 2

The Cochlea Model Chip

A simplified cochlea model considers the cochlea to be a filter bank consisting of a series of bandpass filters. The frequency responses of each of the bandpass filters is a dilation of one model filter, thus the filters maintain a constant relative tuning (or have "constant-Q"). This is a reasonable model of the response of the basilar membrane of the cochlea, though more complex cochlea models may also include effects of fluid-cilia coupling, half wave rectification of ionic channel currents, low pass filtering due to the hair cell membrane, and lateral inhibition.

There have been several examples of hardware implementations of cochlea models which employ analog designs using subthreshold transconductance amplifiers. The approach by Lyon and Watts [4] [5] uses cascaded second-order lowpass sections. The approach by Liu [6] uses a cascade of first-order lowpass sections, followed by a second-order bandpass filters. While successful in many respects, the transconductance amplifier designs suffer from the effects of parasitic capacitances and nonuniformity of the fabrication process, thus requiring post-fabrication tuning to compensate for these effects. They also can have low dynamic range because of the limited linear range of the transconductance

amplifiers.

Lin's VLSI cochlea model [3] is based on switched-capacitor filters (SCF's). SCF's have extremely precise and reliable response characteristics which obviate the need for post-fabrication tuning. The frequency response of SCF filters are dependent on the sampling clock frequency, so that the filter bank can cover different frequency ranges by using different sampling clock frequencies. The filter bank center frequencies can also be interdigitated to functionally increase the number of cochlea channels in a certain frequency range. In Chapter 3 I will show how this method can be used to create a 64 channel cochlea model from two 32 channel cochlea model chips.

One problem with SCF's is the need for very large capacitors, especially for filters with low pole frequencies. To maximize the number of cochlear channels per chip, Lin incorporated techniques to minimize the capacitor spread ratio (the ratio between the area of the largest and smallest capacitors in a circuit). The capacitor spread ratio of conventional SCF biquad filters is $\frac{1}{\Omega_0 T}$, where Ω_0 is the pole frequency and T is the sampling period. Lin used the very large time constant (VLT) biquad of Nagaraj [7] to reduce the capacitor spread ratio to $\sqrt{\frac{1}{\Omega_0 T}}$.

Lin also developed a method of achieving the desired frequency response of a parallel filter bank by making each channel a weighted sum of simple elementary filters, such as biquads [8]. Sharing of the elementary filters by adjacent channels can be accomplished by having elementary filters which are dilations of one particular filter, and the summed output channels will then be dilations of each other. Fortunately, this is precisely what is needed for a cochlea model. The circuit sharing drastically reduces the circuit count required to achieve high-order

responses in a parallel filter bank architecture.

These area-saving techniques allowed Lin to fabricate a 32-channel cochlea model chip with one first order highpass filter, followed by 32 parallel 6th order lowpass filters. The chip die area was 4.6 x 6.8 mm^2 , and was fabricated using the MOSIS 2 μm CMOS process. The filter bank covers the frequency range from 0.2 to 6.4 kHz using a sampling frequency of 125 kHz. The highpass filtering means that while the filter responses all have the same general shape, they are not simply dilations of each other, but the responses of higher frequency channels have higher overall gain.

Due to space limitations, there are no buffer amplifiers on the chip to provide low-impedance outputs. There was also not enough room for a nonoverlapping biphasic clock generator on-chip. Both of these functions had to be achieved by off-chip systems, and will be described in Chapter 3.

Chapter 3

Design of the Interface System

3.1 Design Criteria

The cochlea model interface system was designed with these criteria in mind:

- Near-real-time processing of digitized audio into a cochlear representation
- The ability to process long speech data (greater than 1 second)
- The processed speech data must be made available in the appropriate file format for use in existing multi-channel signal processing software
- Implementation of a 64 channel cochlea model (requiring two 32 channel chips)
- Reasonable cost
- Ease of use

These criteria directed the choice of hardware and software systems throughout the project. The end product was to be a useful tool for rapidly generating cochlear representations of speech for use in experiments, such as models of

primary auditory cortex, speech recognition using neural networks or hidden Markov models, improving speech understandability, and similar projects. The data had to be available over the local network for use on computer workstations by researchers.

3.2 Hardware Design

The interface system hardware design consisted of two main projects. The first was development of appropriate circuitry for analog interface of the SCF cochlea model chips themselves, and the second was the A/D and D/A systems.

3.2.1 Analog Interface Circuitry

To work properly, the cochlea model chips required some support circuitry for proper clocking and analog interfacing. These circuits were developed first during the earliest testing of the newly fabricated cochlea model chips.

During design of the cochlea chips, it was hoped they could operate using +/- 5V supply voltages. A BK precision triple output power supply was used to provide the proper supplies. The BK supply also had a very finely tunable current limitation setting, which is important when dealing with latchup-prone CMOS circuits. At +/- 5V, the cochlea chip draws greater than 300 mA, and runs dangerously hot. With the supply voltages lowered to +/- 4V, the cochlea chip only draws 220 mA, and ran at an acceptable temperature. The op-amps of the SCF filters are the primary cause of dissipated power in the cochlea chips. With just over 100 op-amps on the chip, there is little room for power hungry op-amps. Future version of the VLSI cochlea chip will need to be re-engineered

with op-amps that draw less current so that the supply voltages can be raised, and thus the dynamic range of the analog signals can be increased.

Since the cochlea chips were based on switched capacitor filters, they required appropriate clocking mechanisms. To implement the 64-channel filter bank, slightly different clock frequencies were needed to be available to the two VLSI cochlea chips. Figure 3.1 shows the architecture of the reference clock generators.

The master reference frequency is provided by a CMOS clock generator package, operating at 4 MHz. The squarewave from the generator is divided down by two 74HC161 4-bit presettable binary counters. The counter presets are selected by an 8 position DIP switch for flexibility. The counter outputs are then converted to a symmetric squarewave using a 4013 dual D flip-flop, and finally buffered to the cochlea chip boards through a 4009 hex inverting buffer. For most testing runs, the DIP switches were set to 0 and 1, dividing the 4 MHz clock by 16 and 15, respectively, resulting in sampling frequencies of 125.4 kHz and 133.8 kHz (the process of converting the counter carry pulses to a symmetric squarewave involves a further division by two of the reference clock frequency).

Each of the cochlea model chips required an off-chip biphasic nonoverlapping clock generator. This circuit develops the two complementary logic signals (ϕ_1 and ϕ_2) required for the CMOS transmission gates which act as switches in the cochlea chips. A nonoverlapping clock generator from [9] was used, and is shown in Figure 3.2.

In early experimentation, it became evident that the VLSI chip op-amps were not capable of driving the impedance of many off-chip sensors. Therefore LM347 quad op-amps in unity gain configuration were used to buffer the chip outputs.

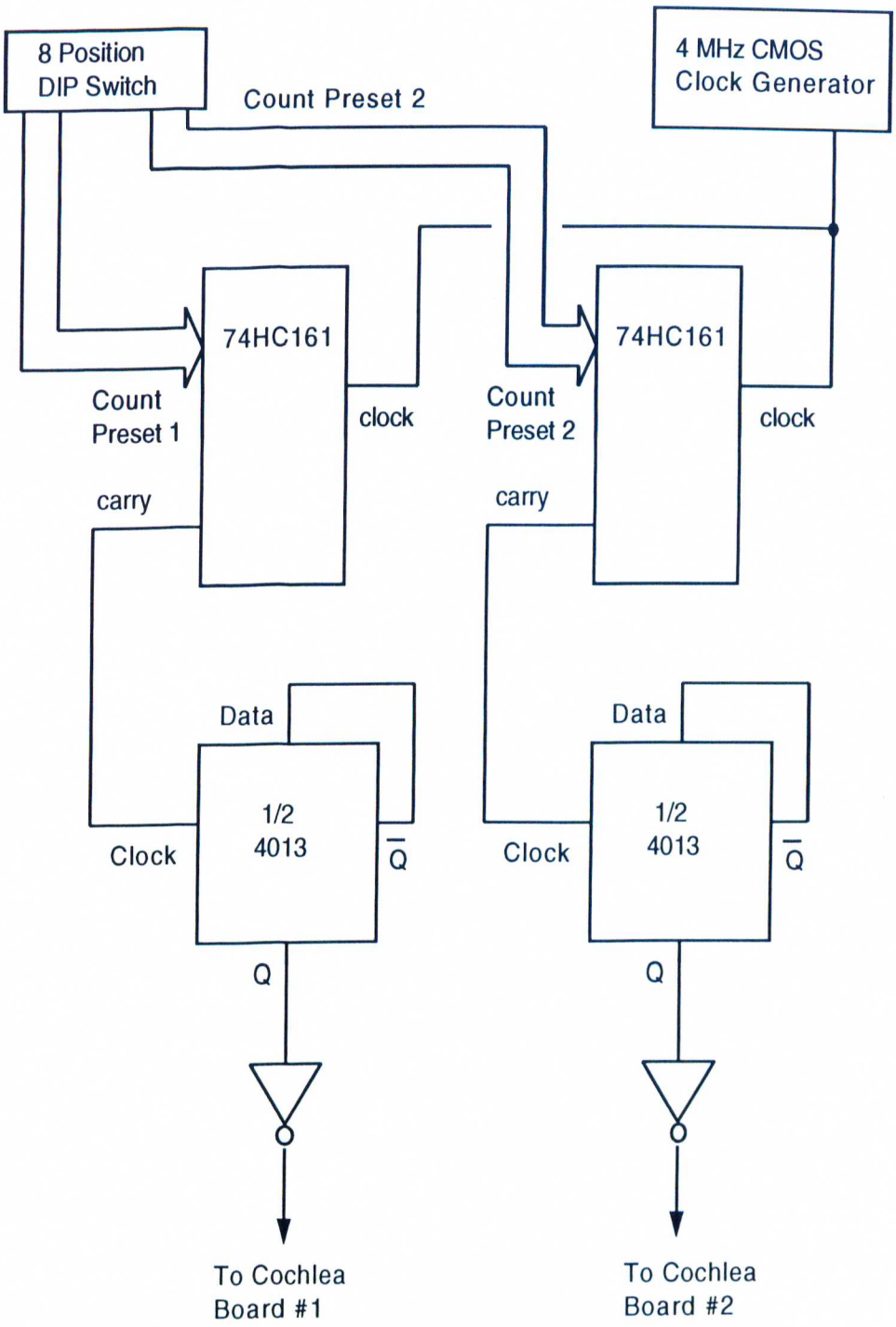

Figure 3.1: Reference Clock Generators

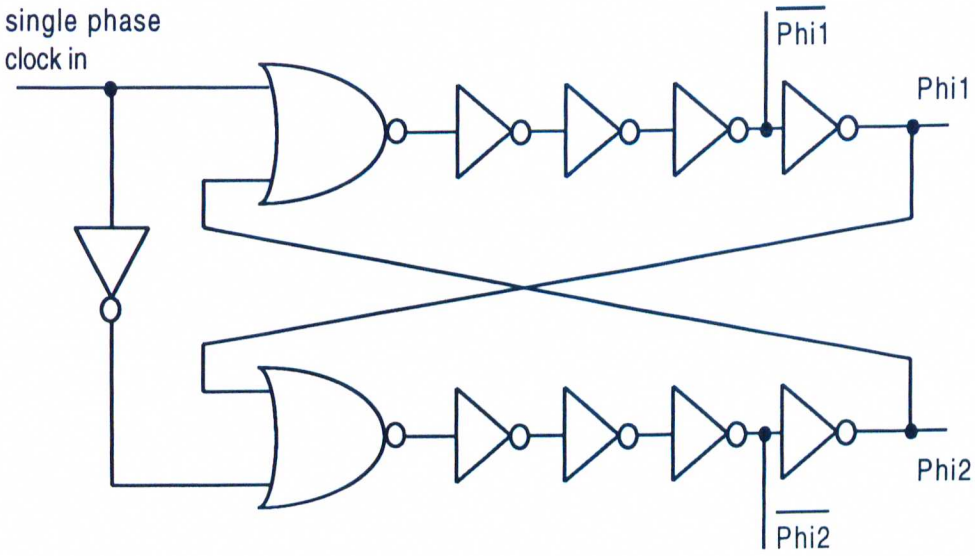

Figure 3.2: Nonoverlapping Clock Generator Circuit

3.2.2 A/D and D/A Interface

The requirement for near-real-time cochlear processing meant that the A/D system would have to be very fast to read all 64 channels at a reasonable sampling rate. It was decided a good compromise would be to only require 16 channels to be read at any time. All 64 channels would be read by playing the sampled speech data four times and recording from a bank of 16 channels each iteration. Although this expanded the processing time to at least four times the actual length of the speech data, it was still felt that this would not significantly offset the speed advantage of the analog VLSI cochlea model when compared with computer models.

There was significant effort put into deciding on the proper A/D system to utilize. The A/D system had to have the following characteristics:

- at least 16 A/D channels

- per-channel sampling of at least 20 kHz
- capable of being operated from user programs
- at least 8 bit A/D resolution
- compatability with lab computers (HP Vectra PC, SparcStation IIPX, or HP 700/X)

The D/A system to play the digitized speech signal had to have the following characteristics:

- at least one D/A channel with 20 kHz or faster sampling rate
- enough extra analog or digital outputs to provide multiplexing control for the channel banks
- at least 8 bit D/A resolution

It turns out that there are few products on the market that are of reasonable cost which address analog data acquisition for a large number of channels at the required sampling rates to adequately represent speech signals. One board which fulfilled both the requirements of A/D and D/A systems is the WB-FLASH12 from Omega. Table 3.1 shows its specifications.

Because only 16 analog channels are read at a time, it was necessary to design a 4-way multiplexing system. The outputs of the analog output op-amp buffer chips were gated by 74HC4066 quad analog switches. TTL level signals from the WBFLASH-12 board were converted to +/- 4V CMOS switch control logic through the use of pull-up resistors and op-amps used as comparators.

A/D channels	16
maximum sampling rate for each channel	62.5 kHz
A/D throughput	1 MHz
A/D memory	1M sample
A/D resolution	12-14 bits
TTL digital I/O lines	8
D/A channels	2
D/A sample buffer	256k samples
Computer system	IBM-PC clones

Table 3.1: Specifications of WB-FLASH12 Card

Figure 3.3 shows the simplified block diagram of the data acquisition system. Digitized speech from the HP Vectra PC is converted to analog speech by the WB-FLASH12 D/A system. The analog speech is low pass filtered by the Krohn-Hite 3905A Multichannel Filter (7th order elliptic, cutoff frequency is 10 kHz). This step smoothes out the high frequency digital sample noise of the speech signal, and prevents this noise from being aliased into the desired speech signal by the sampling action of the SCF's. The speech is then applied to the VLSI cochlea boards, and one of the four 16-channel banks is selected to be output from the cochlea boards. Each board has one 32 channel cochlea chip, and the two banks of 16 channels from each chip comes from its even and odd output channels. The 16 analog outputs are then smoothed by first-order low pass RC filters (R is 200Ω, C is 0.05 μF, cutoff frequency is 15.91 kHz). The resistance of the filters was kept low to avoid any potential resistor divider voltage drops

due to finite input impedance of the A/D system. The filtered analog output is then digitized by the WB-FLASH12, and the digitized data is sent to the HP Vectra PC. For most speech applications, the A/D sampling period was $3\mu s$ per channel, or $48\mu s$ for all 16 channels, resulting in a 20.83 kHz sampling frequency. This frequency was a good compromise between adequate sampling of the audio data and minimizing the size of data files. All A/D sampling was done using 12 bits resolution.

3.3 Filter Characterization Test Equipment

The HP3566 PC Spectrum/Network Analyzer, already available in the NSL lab, was used to do simple filter characterization of the VLSI cochlea model. The HP3566 Analyzer interfaces with the HP Vectra PC via a HP-IB bus connection. The analyzer was used to determine the frequency response of some of the cochlear channels. The time capture mode of the analyzer was also valuable in debugging the data acquisition system.

3.4 Data Acquisition Software

3.4.1 Software Design Criteria

The data acquisition software written on the HP Vectra PC had to be capable of the following functions:

- record digitized audio samples for later processing by the VLSI cochlea model system
- playing digitized audio samples for previewing before processing

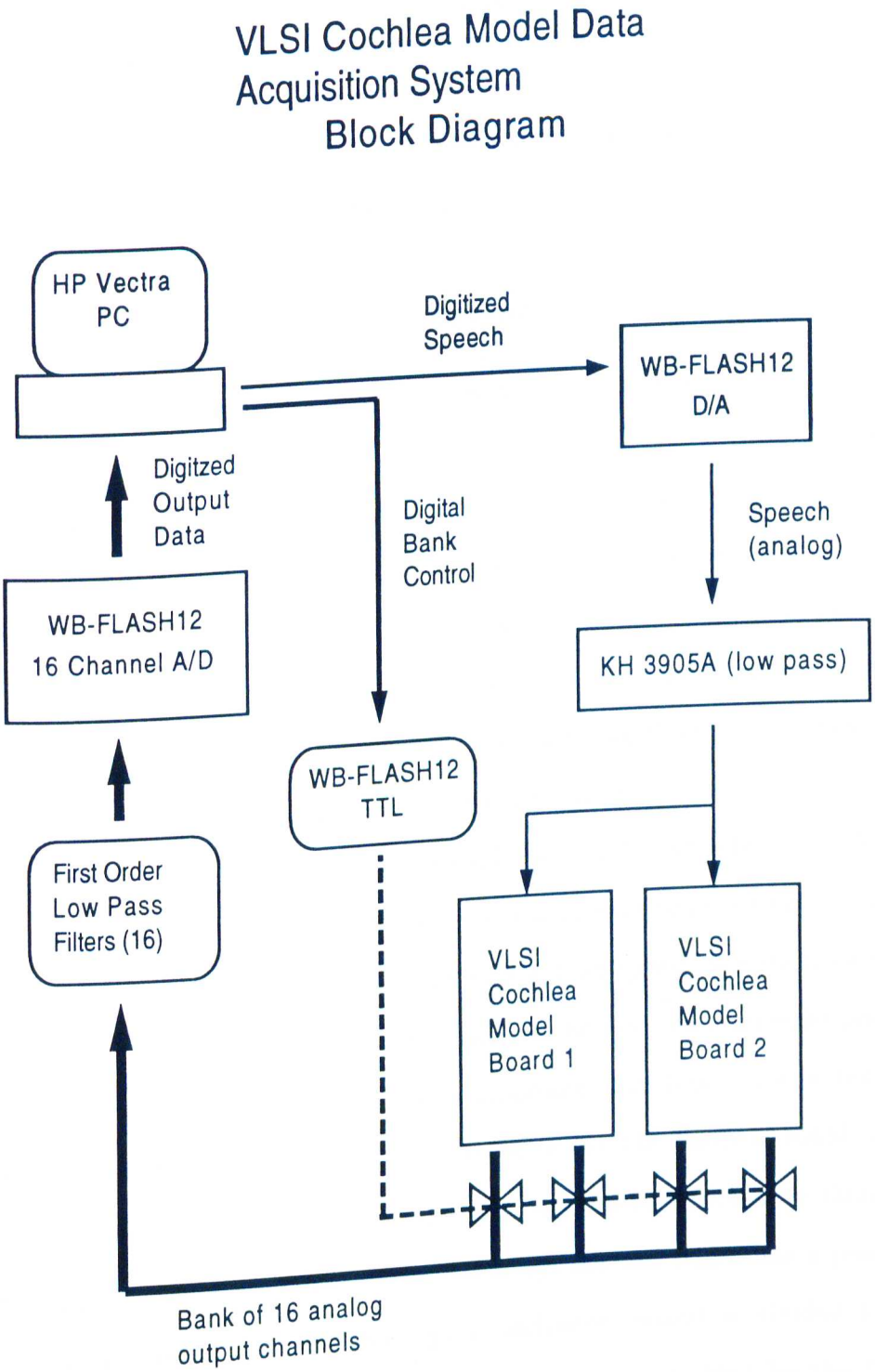

Figure 3.3: Data Acquisition System Block Diagram

- processing digitally sampled audio data through the VLSI cochlea model system
- have an on-screen "demo mode" which provided a real-time graphical demonstration of the cochlea model

3.4.2 WBFLASH-12 Description

The WBFLASH-12 can be accessed by a software driver from BASIC, PASCAL, Fortran, C, assembly, and ASYST languages. The driver, however, does not provide all of the functionality of the WBFLASH-12 card. In particular, it is impossible to perform both D/A and A/D conversions at the same time using the driver. Therefore the card had to be programmed directly, which was only supported by using the C language. Borland C++ was the particular programming development package used. The program for performing all the above mentioned data acquisition and processing tasks is called **cochint.c** .

The WBFLASH-12 card is based around the Z8536 controller chip. Analog data coming into the board goes through a multiplexor to select one of the 16 input channels. The input is then feed into a high-speed, precision switched gain differential amplifier. The amplifier gain can be selected from eight possible gains, from times one to time 200. Once amplified, the data is converted by a 1 MHz ADC, and data is automatically stored in the 1M words of RAM on the daughterboard. The daughterboard also has two DACs, which have their own 256K data buffer. To accurately time measurements, the board has a precision frequency synthesizer. A 16 MHz crystal oscillator output is divided by the "reference divider." A 8-16 MHz VCO has its output divided by the "VCO divider." The outputs of these two dividers are applied to a phase detector, and

the resulting voltage is fed back to the VCO, forming a phase-locked loop. The output of the VCO is then divided down by 16, and can be divided down further if needed. This provides an extremely stable and accurate clocking signal. The WBFLASH-12 is also capable of triggered recording. One important fact to note is that the D/A system runs at a rate eight times as fast as the A/D system. In **cochint.c**, when a user is prompted for "length" and 'offset," the number of D/A samples is meant. This makes it easier to discuss which part of a digitized speech sample you wish to process.

3.4.3 Initialization

Cochint.c begins by initializing the WBFLASH-12 main Z8536 and daughter-board Z8536. The phase locked loop dividers are then set up (both reference and VCO dividers are set to 256), yielding a default sampling frequency of 1 MHz. Later in the program, the user can change sampling frequencies. The calibration data stored in an EEPROM (which has been previously set up) is downloaded to properly calibrate the D/A and A/D systems. The title page is then displayed.

Once a key has been pressed from the title page, the program displays the main menu of options. Figure 3.4 shows the flow diagram of the program from the main menu.

3.4.4 Recording Data

Before digitized speech can be processed by the VLSI cochlea model, it needs to be recorded. **Cochint.c** allows the user to do this using the WBFLASH-12 card. It is possible that for certain experiments, pre-digitized speech corpuses like TIMIT may be desired, but for testing of the data acquisition system all

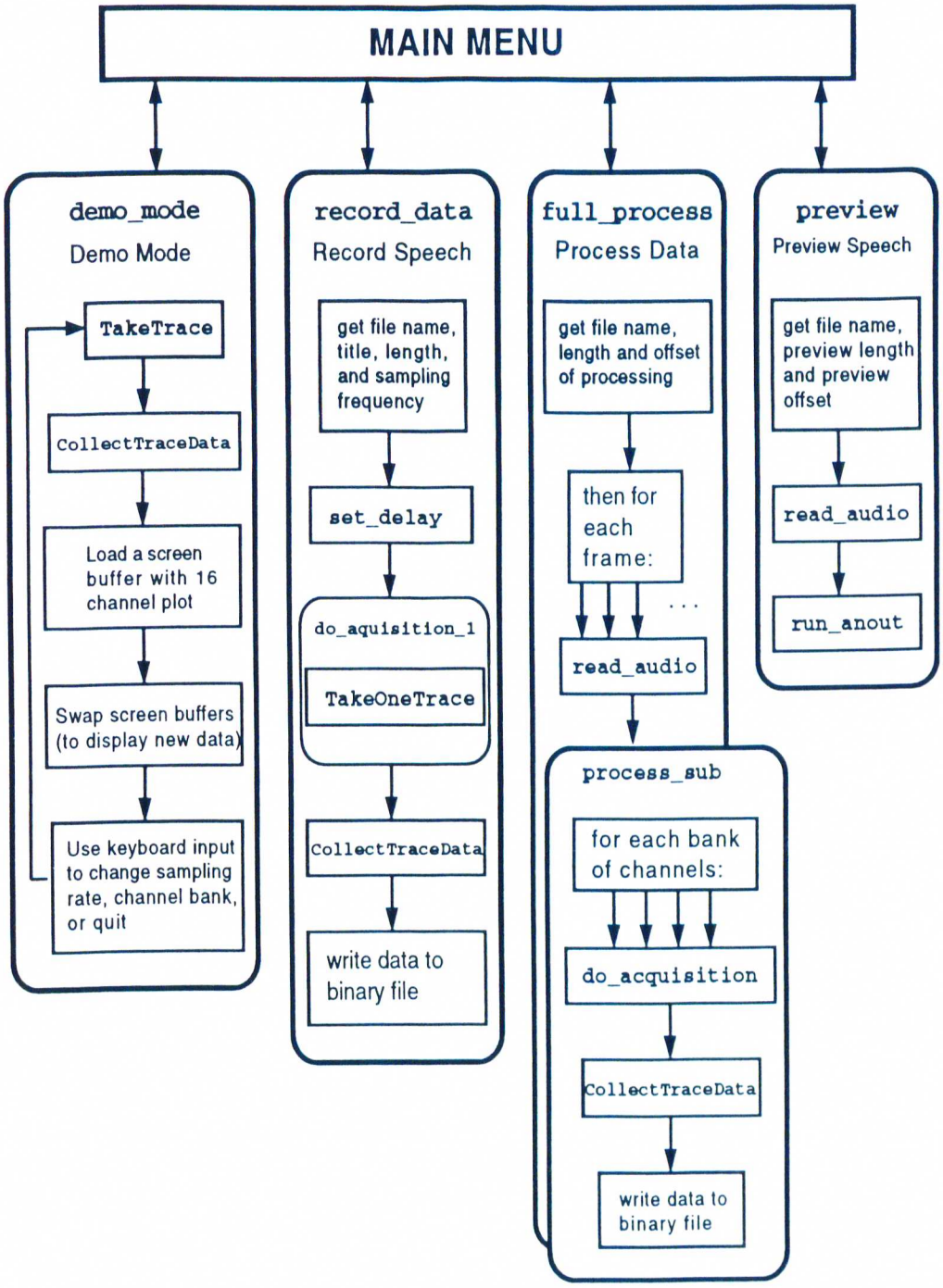

Figure 3.4: cochint.c program flowgraph

speech was digitized using **cochint.c**.

The routine which records data is **record_data**. It queries the user for the sampling period (in ns), number of data points, title of data, and the file name to store it as. The data files are stored with a suffix of .dat in binary format, the header files, with title, length, and sampling frequency, are stored with a suffix of .hed in ASCII format. The audio data files of the acquisition system are of binary type to minimize required disk space and read/write times. The sampling frequency is set with the **set_delay** function, the "Ready to Record" message it put on the screen, and the user is asked to hit any key to start the recording. This allows the user to get ready to begin talking into a microphone or to start a tape.

Do_acquistion_1 is called by **record_data**. **Do_acquistion_1** records data from only one channel, allowing for recording of up to 1M samples. For most reasonable choices of sampling frequencies, this will allow for up to 6 seconds of speech. Longer files can be constructed by concatenating recorded binary data files and using an editor to update the length noted in the header file. **Do_acquistion_1** begins by setting the trigger to a small DC value to help eliminate long pauses at the beginning of the sampling, and then records the data using the **TakeOneTrace** function (supplied by Omega) which actually commands the Z8536 to run and perform the recording.

Record_data then must write the samples to disk. It uses the **CollectTraceData** function (also supplied by Omega) to transfer data from the WBFLASH-12 card to the PC RAM in 8192 sample blocks, and it writes each of those blocks to a binary file. Finally, it closes the data file and returns to the main menu.

3.4.5 Previewing data

It is important before processing a speech segment that it is certain that the proper audio data was recorded. The **preview** function allows the user to do that. **Preview** asks the user for the file name, and then loads in and displays the length and title from the header file. The user is then prompted for offset and length of the audio selection to play. The offset is the offset in samples from the beginning of the file, and the length is how many samples to play. This allows the user to be more selective about which part of the recorded speech to play, for instance, if individual isolated words are to be selected from a sentence.

Once the details of the file to be played are entered, the data is read into the daughterboard D/A buffer using **read_audio**. Data is read in blocks of 8192 samples, and stored in board RAM using the **LoadDACBuffer** command provided by Omega.

The sampled data is then played by the **run_anout** command which sets up the DAC control registers using **dac_ctrl**, and commands the Z8536 to run and play the sampled data. The data is played over and over until the user hits a key to stop it. Once a key is hit, the user is returned to the main menu.

3.4.6 Processing data

Processing the sampled speech through the cochlea model can be thought of as a combination of previewing and recording speech data, except instead of only recording 1 channel, 16 are recorded from. The function which performs processing of sampled speech data is **full_process**.

The user is prompted for the digitzed speech data file name, the header file is read in, and title and length are displayed. The user enters both initial offset

and number of samples to process. Again, the use of offset and length allows for a particular part of the sampled data to be processed.

Read_audio is called to load in the first block of speech data. **Process_sub** is then called, and it in turn calls **do_acquistion** four times, once for each bank of 16 channels. **Do_acquisition** is the 16 channel non-triggered version of **do_acquistion_1**, and calls the Omega provided routine **TakeTrace** to collect analog data store it in board RAM. **Do_acqusition** selects the proper channel bank by calling **OutputDigitalByte**, provided by Omega, which sets up the proper 2 bit TTL bank select code. **Process_sub** then calls **CollectTraceData** and the processed 16 channel data is written out to the output binary file (with a .out suffix) in 8192 byte blocks.

Read_audio and **process_sub** are called again for the next "frame" of data, each time taking 15,000 data points, until all of the required data has been processed. Because of the break between frames, we have to do some processing to make up for the step response of the cochlear filters which is superimposed on the output from the speech data. Because the step response is finite, we need only replay a small portion of the data which was at the end of the last frame at the beginning of each new frame. This is similar to the "overlap-save" DSP method mentioned in [10]. The length of data to be replayed depends on the step response of the slowest filter, and 5 ms has been found to be quite adequate.

3.4.7 Demo Mode

The "Demo Mode" displays 16 cochlea channel outputs in real-time to the HP Vectra PC screen. It calls **TakeTrace** to collect analog data into the board RAM, then calls **CollectTraceData** to bring the data over the bus to the Vec-

tra RAM. The screen is painted with the analog waveforms from one bank of channels, and this process repeats until the user types "q" to quit. Double-buffering of the graphics screen is used to eliminate flicker.

The user also has the choice to increase or decrease the sampling period using the "+" and "-" keys. A different bank may be selected for display by pressing the keys 0-3.

Besides being an interesting demonstration, the "Demo Mode" is also a useful diagnostic tool to identify damaged cochlea chip channels, or bad connections between the chips and the WBFLASH-12 board.

3.5 Network Data Transfer

Before the processed audio data can be made available to digital signal processing software, it must be transfered over the lab network from the HP Vectra PC to one of the Sun Sparc workstations. An Intel EtherExpress 16 card was used to provide thin ethernet communications for the Vectra PC. Public domain CUTCP FTP from Clarkson University was obtained over the internet for TCP/IP data transfer, and also CUTCP TELNET was obtained to allow for interactive login sessions with TCP/IP to the Sun Sparc machines. The public domain **pdclksml** package was also obtained to set the Vectra PC clock from the local time server so that files written by the **cochint.c** software would have the correct time and date.

3.6 Data Conversion Software

The raw data coming from the IBM-PC clone must go through some additional processing on the Sun Sparc before it is in a form usable by Entropic Signal Processing System 3 (ESPS) [11] software from Entropic Speech, Inc. ESPS is a package of interacting signal processing programs. It provides utilities for many types of common signal processing operations within the UNIX operating environment. Instead of one large signal processing program, there are many small ones, and they may be linked together with pipes and I/O redirection from the UNIX shell.

The processed audio data from the VLSI cochlea model is converted to the ESPS FEA (arbitrary feature) format. The FEA file type is for general purpose use for files with fixed-length records. It can be used by the many of the ESPS programs, including the **waves+** program [12], an interactive graphics interface to ESPS. FEA files can also be converted for use by the **x3dplot** program developed in the Neural Systems Lab.

The data conversion software is written in shell script and C. The **process** shell script runs the various C programs required to convert the data.

The first conversion involves changing the data from the IBM-PC binary format to the Sparc binary format. The FTP binary transfer adds extra bits to make up for the larger word size on the Sparc. These extra bits are stripped away by the program **dataconv.c**.

The data coming from the IBM is still stored in blocks of 16 channels. Due the the large size of the processed audio data, re-shuffling the data back into 64 channel format would be very slow on the IBM. Therefore it is performed on the faster Sparc by the **slice.c** program. Figure 3.5 shows how the data from the

IBM is properly arranged into 64 channel format. The bank shuffle stage puts together all the data from each particular channel bank into a single file for each bank. The channel shuffle stage reads out one channel at a time from each of the bank files in turn, and writes that data to disk. Then the data is in groups of all 64 numerically ordered channels, and all the groups of 64 are in proper time sequence.

The ESPS program **addfeahd** adds the proper header information to create a FEA file, readable by **xwaves**, the X Windows version of **waves+**.

The shell script **doplot** transforms the FEA file to a format readable by **x3dplot** by using the **convert** program which converts the type SHORT binary data to FLOAT. **X3dplot** plots the multichannel data in an aesthetically pleasing manner with constant scaling for all channels.

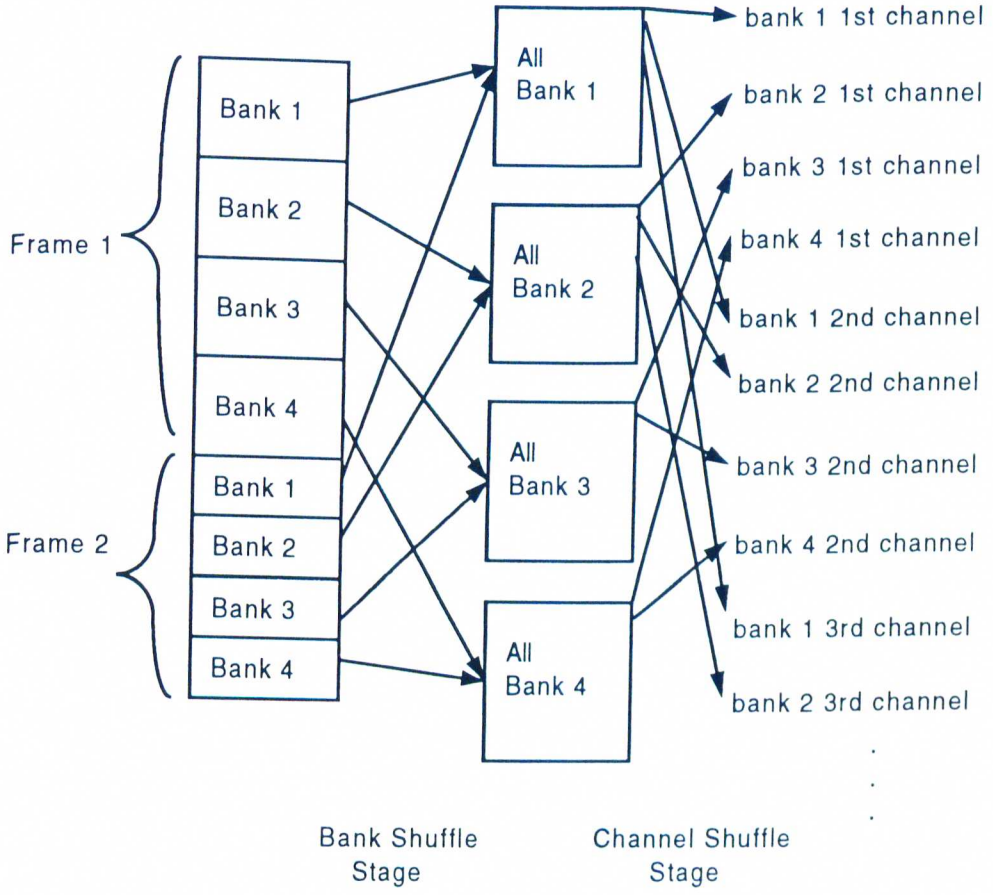

Figure 3.5: Slice.c data reshuffling

Chapter 4

Results and Conclusions

4.1 Frequency Response of the Cochlear Channels

The HP3566 PC Spectrum Analyzer was used to measure the frequency response of cochlear filters. The general shape of the filters agreed with those described in [3]. The frequency magnitude response of channels 34, 59, and 64 are shown in Figure 4.1. The similar shape of the cochlear filters is evident, and they are only differ by a dilation and a high-pass pre-emphasis.

4.2 Single Sinusoid Tests

Single sinusoids at 200 Hz (Figure 4.2), 800 Hz (Figure 4.3), 1.5 kHz (Figure 4.4), 2 kHz (Figure 4.5), and 3 kHz (Figure 4.6), were processed by the cochlea model system. The sinusoids were created from a sinewave sampled data file generator program, **sine.cpp**. The sinusoidal data had a sampling period of $3\mu s$ per channel, and were of length 1000. Data was recorded on the Vectra PC, and then

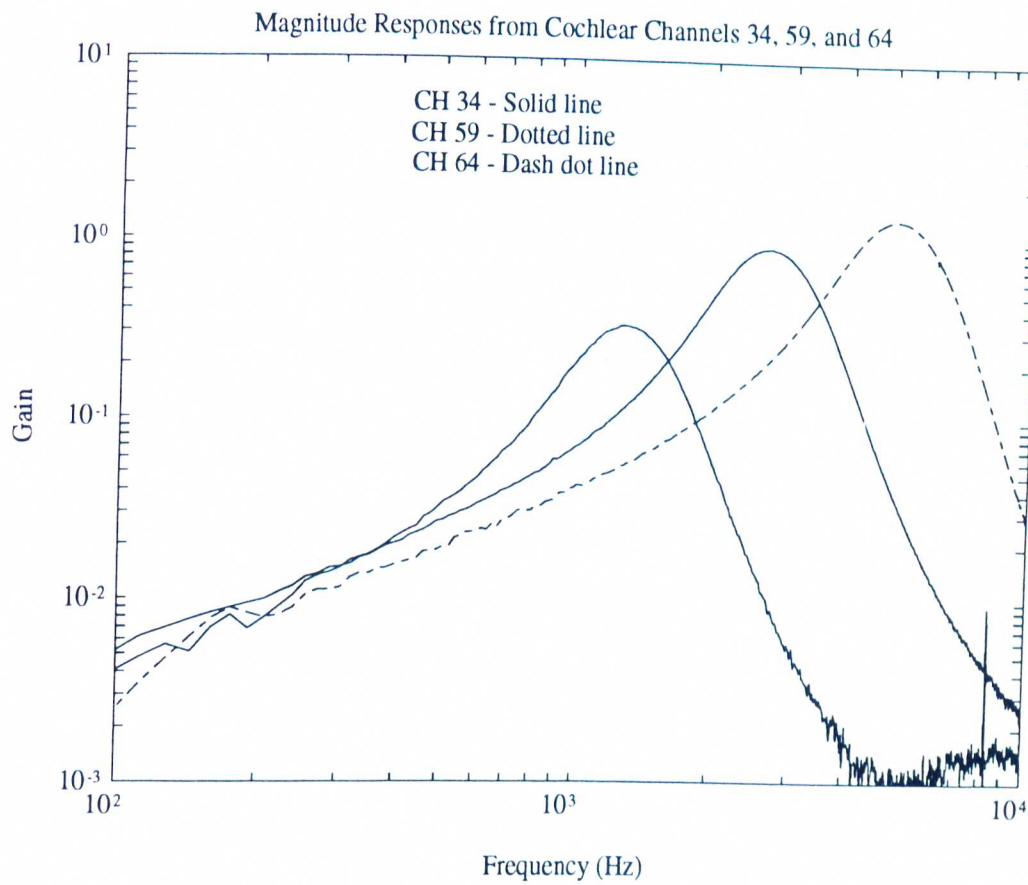

Figure 4.1: Magnitude response for channels 34, 59, and 64

ftp'ed across the lab network to a Sun workstation, where the conversion programs were run, and **x3dplot** was used to plot the data. The higher frequency channels on all of these plots created by **x3dplot** can be found on the bottom of the graph, and the lower frequency channels can be found at top. The system responded as expected, with low frequency channels responding best to low frequencies, and high frequency channels responding best to high frequencies.

4.3 Two Tone Test

A signal consisting of a 2 kHz sinusoid added to a 600 hz sinusoid was processed through the system (Figure 4.7). The cochlear output of another signal consisting of a 1 kHz sinusoid added to a 3 kHz sinusoid is shown in Figure 4.8. The signals were generated by a the **tt.cpp** program, and had the same sampling period and length as the single sinusoids. The VLSI cochlea model's ability to discriminate the two different tones is evident in both examples.

4.4 Single vowel tests

The three vowels that form the "vowel triangle" when first versus second formants are plotted have been recorded and processed through the VLSI cochlea model system. These are the vowel /iy/ as in "beet" (Figure 4.9), the vowel /oo/ as in "boot" (Figure 4.10), and the vowel /A/ as in "hot" (Figure 4.11). Examining plots of responses from the cochlea model channels clearly reveal their formant structure (the formant frequency information in this thesis comes from [13]. It is important to keep in mind that these formant frequencies can vary from person to person.)

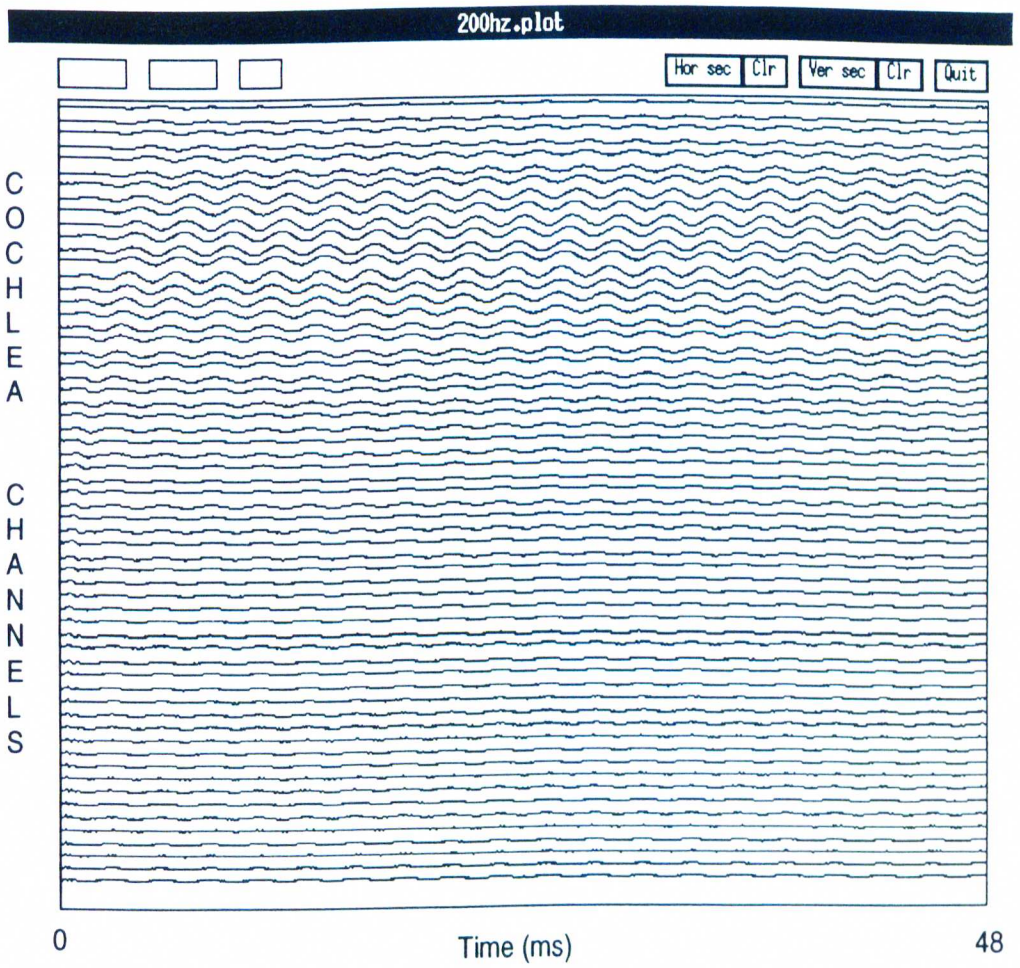

Figure 4.2: Response to single 200 Hz tone (Total time 48ms)

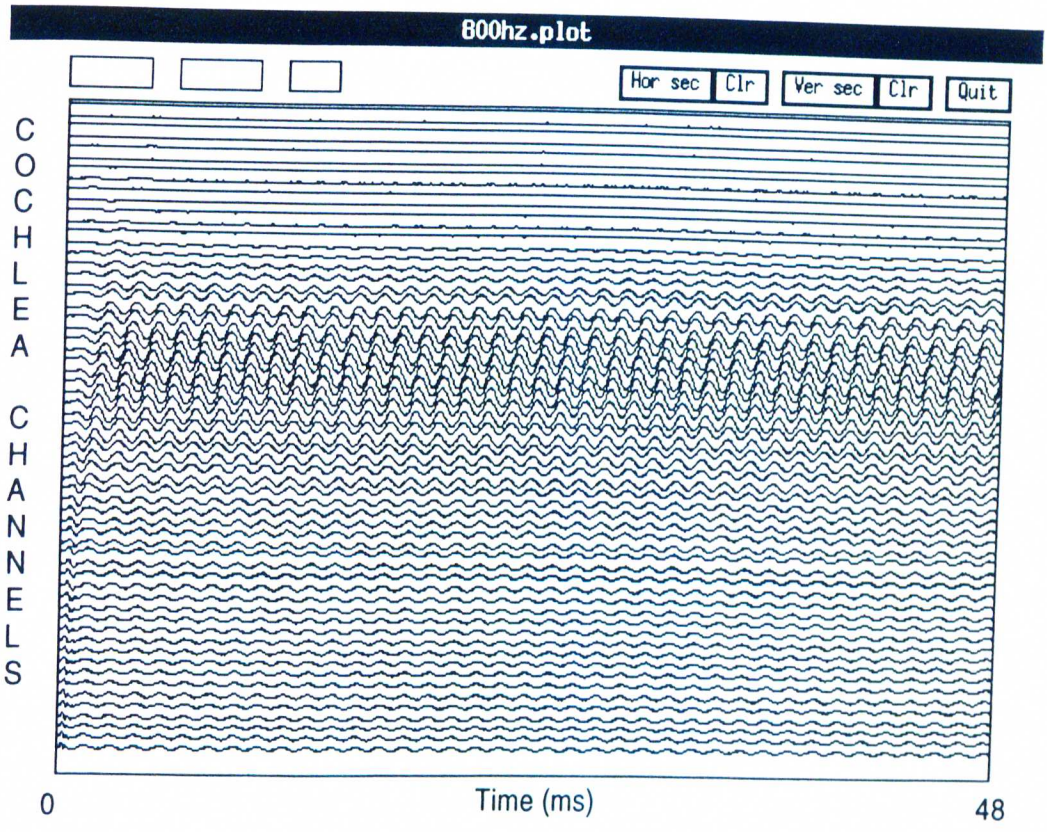

Figure 4.3: Response to single 800 Hz tone (Total time 48ms)

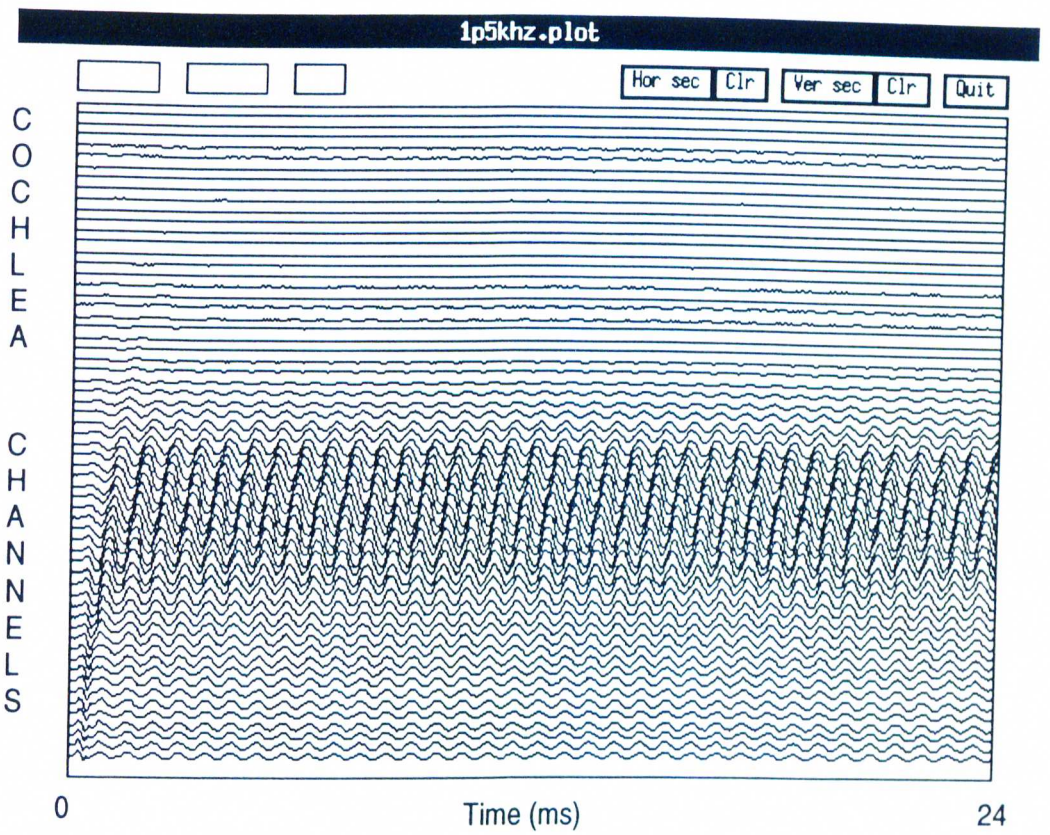

Figure 4.4: Response to single 1.5 kHz tone (Total time 24ms)

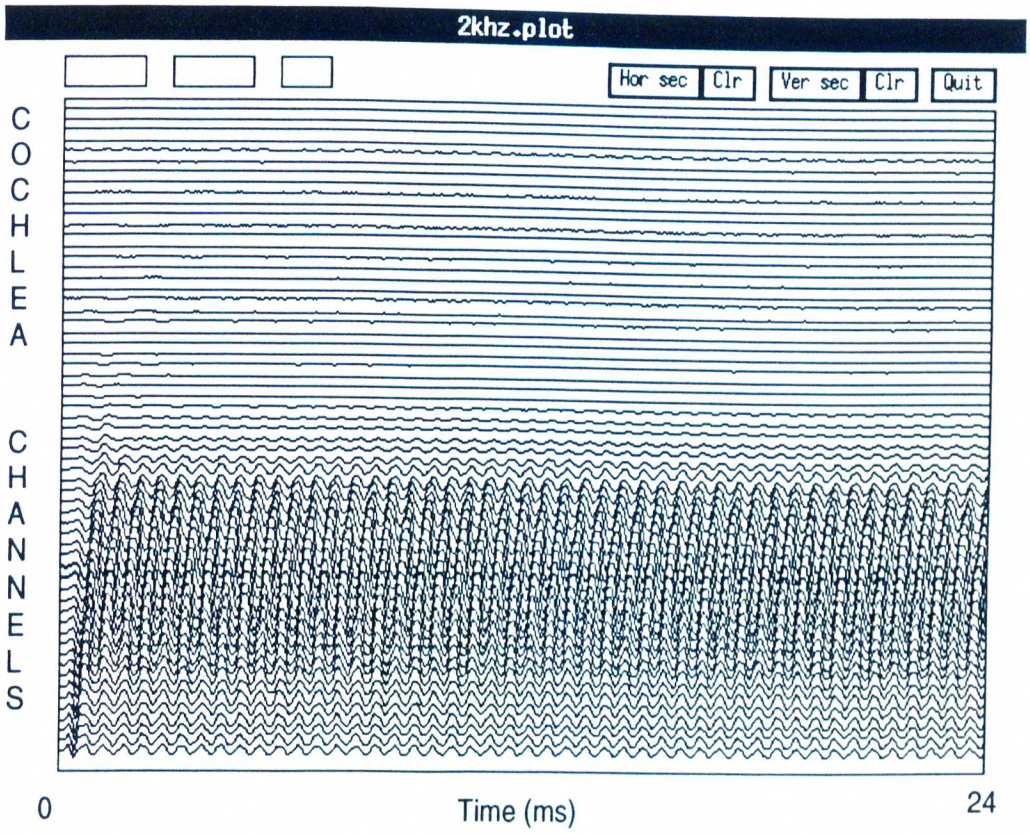

Figure 4.5: Response to single 2 kHz tone (Total time 24ms)

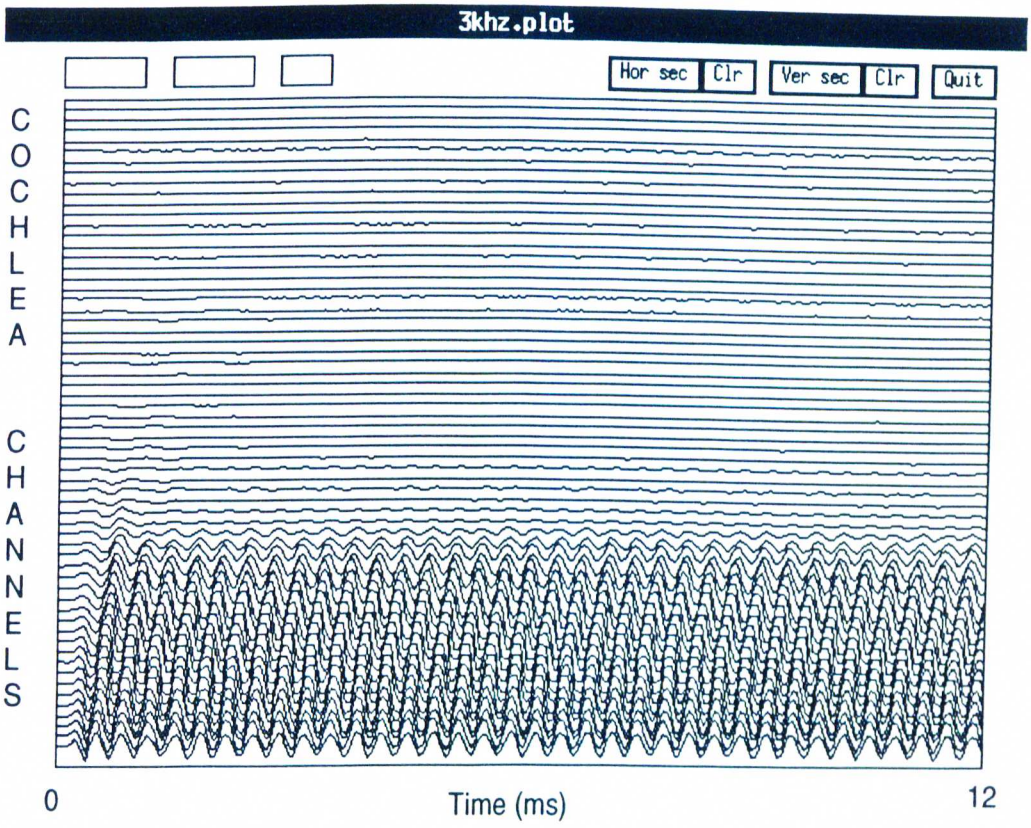

Figure 4.6: Response to single 3 kHz tone (Total time 12ms)

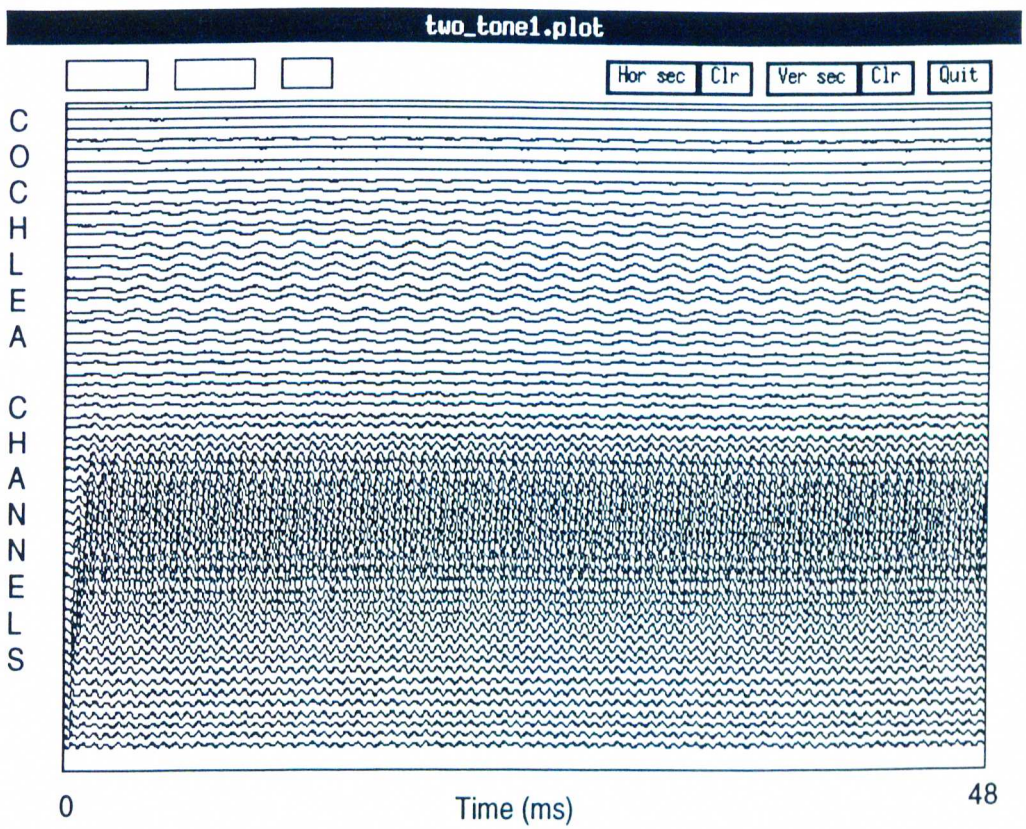

Figure 4.7: Response to 600 Hz tone added with 2 kHz tone (Total time 48ms)

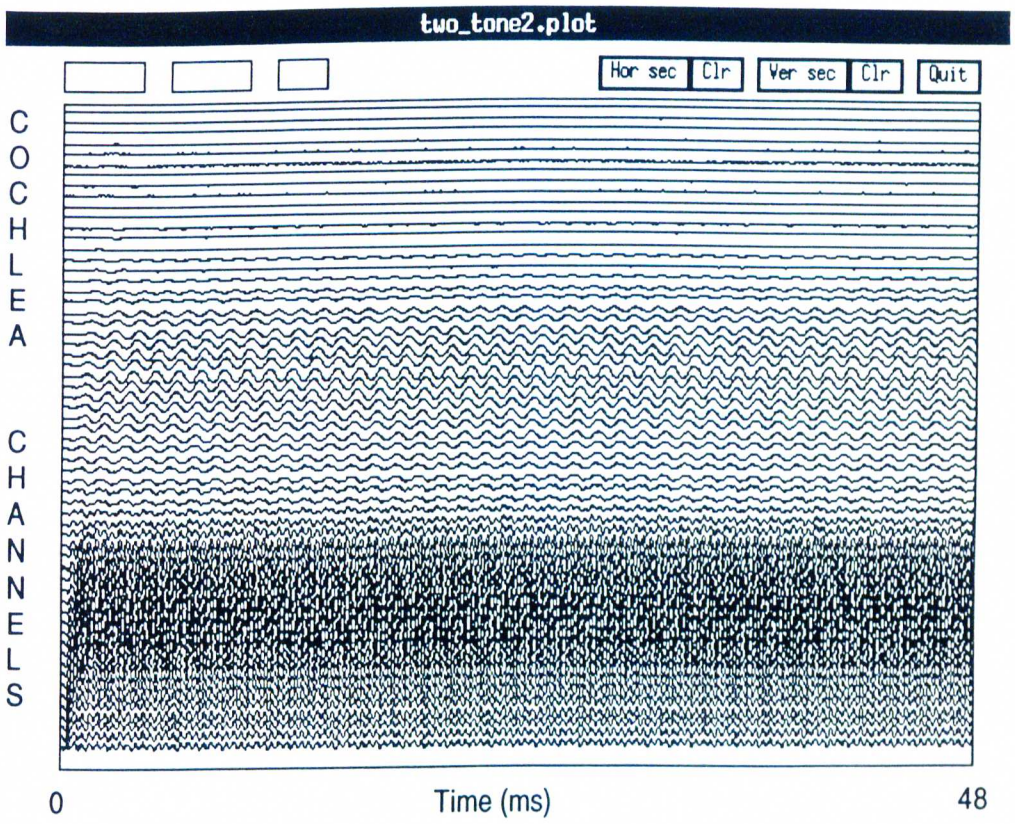

Figure 4.8: Response to 1 kHz tone added with 3 kHz tone (Total time 48ms)

The vowel /iy/ has a very low frequency first formant (270 Hz), and a very high frequency second formant (2290 Hz). From the plot we can make out these two formants, as well as the contribution from the third formant at 3010 Hz. It is important to keep in mind that the high frequency components of speech are highly emphasized by the shape of the cochlear filters, so even the small third and higher formants are visible.

The vowel /oo/ has a low frequency first and second formant (300 Hz and 870 Hz.) In the plot, we can see that most of the cochleal response is in the lower frequencies, and it is possible to discriminate between the two low frequency formants.

The vowel /A/ has a higher frequency first formant than the other vowels (730 Hz), and a second formant at 1 kHz. The effects of these two formants mix in some of the channels between 730 Hz and 1 kHz. But the channels that best respond to the two formants show a more pure response. The third formant at 2440 Hz is also evident.

4.5 Isolated Words

The isolated words, "you," "free," and "bay" were processed by the VLSI cochlea model.

The word "you" contains a short semivowel glide /y/, and the dipthong /ju/ which has a constant first formant near 350 Hz and sweeps the frequency of the second formant from 2.5 kHz to 1.25 kHz. In the Figure 4.12, the formants and their movements are clearly seen. This speech segment lasts 0.48 s.

The word "free" (Figure 4.13) contains the unvoiced fricative /f/, the semivowel

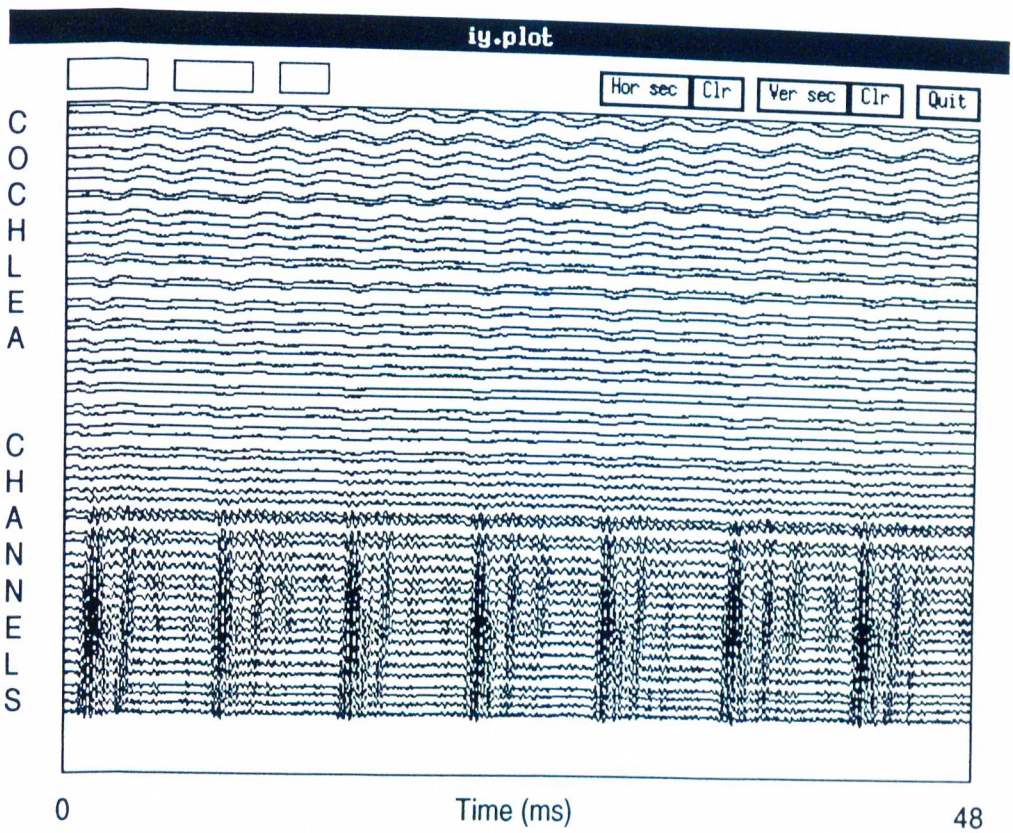

Figure 4.9: Response to the /iy/ vowel sound (Total time 48ms)

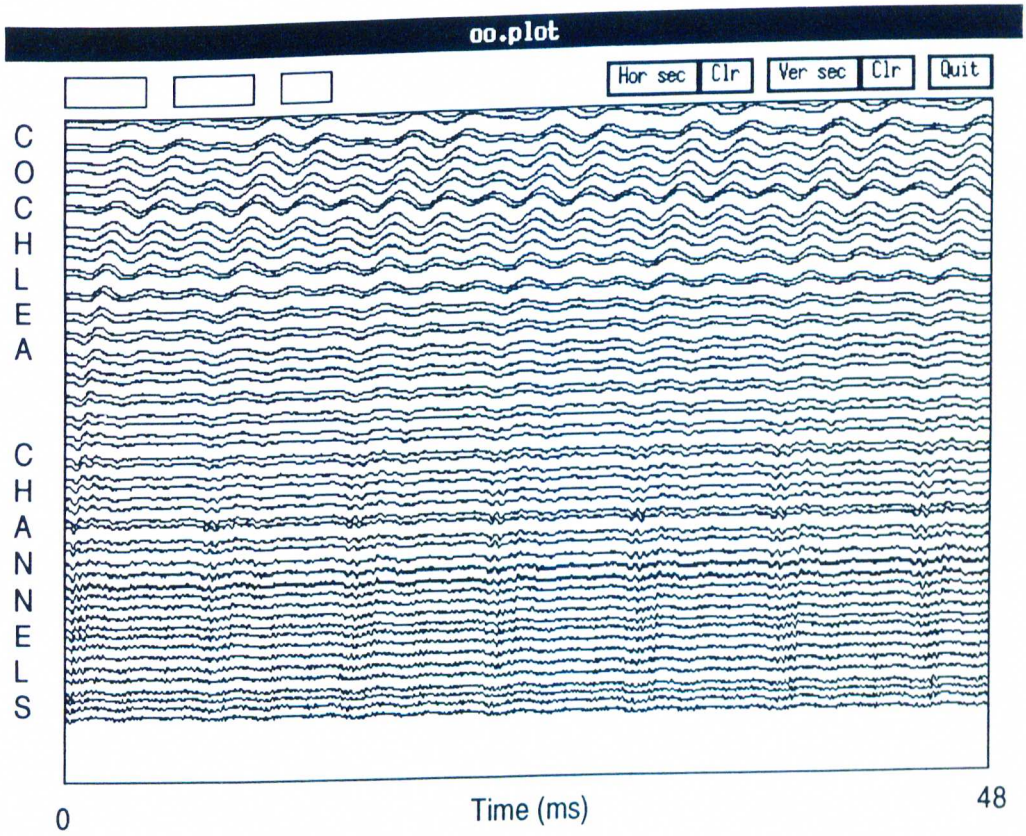

Figure 4.10: Response to the /oo/ vowel sound (Total time 48ms)

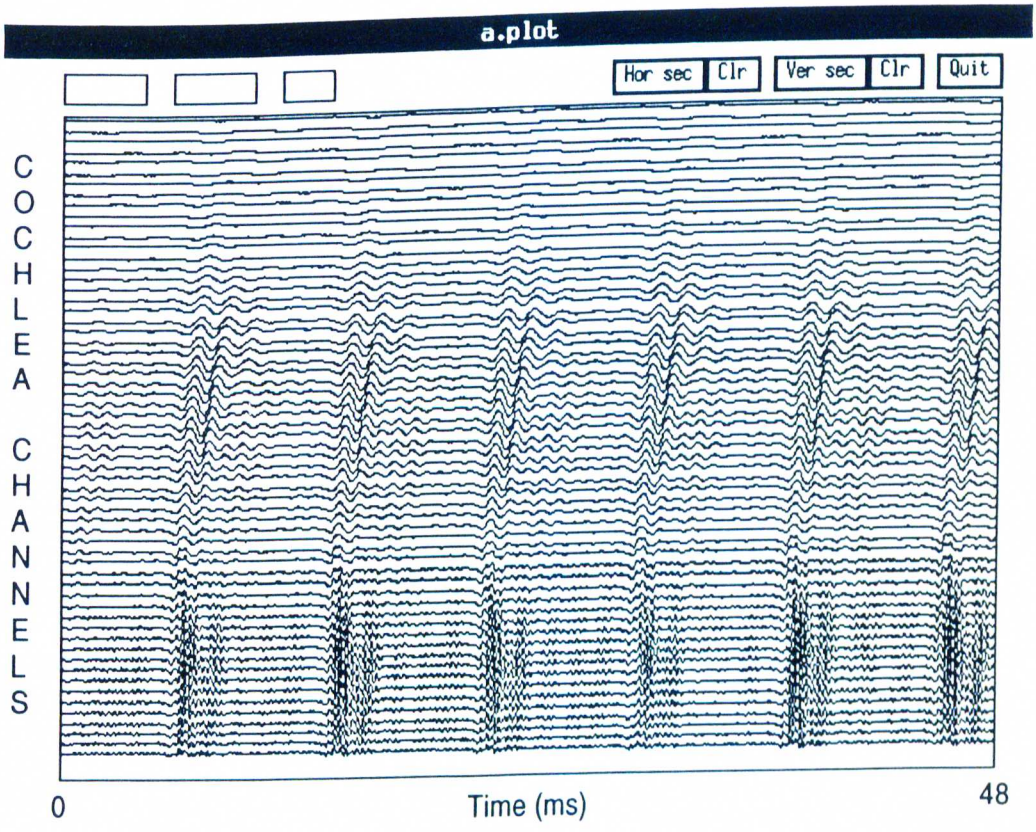

Figure 4.11: Response to the /A/ vowel sound (Total time 48ms)

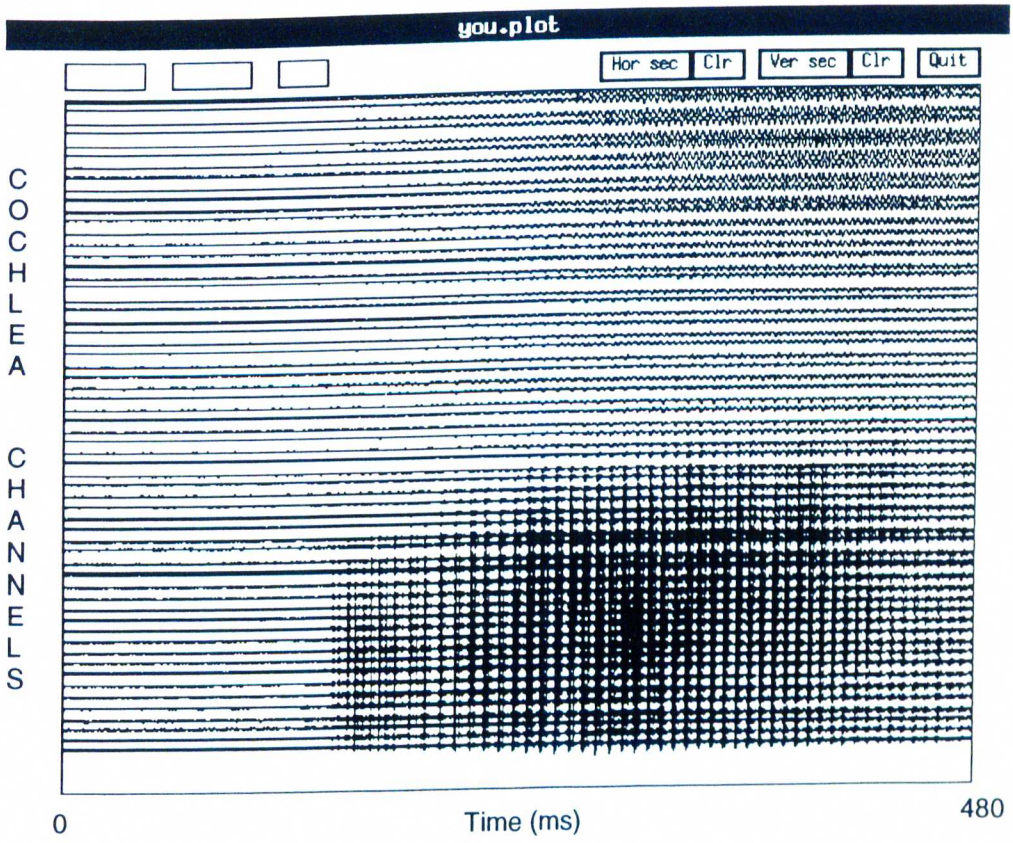

Figure 4.12: Response to the spoken word "you" (Total time 480ms)

glide /r/, and the vowel /iy/. The high frequencies of /f/ are noticeable at the beginning of the plot, and then the dominant glide of /r/, leading into the vowel /iy/ with its high second formant and low first formant. This speech segment lasts 0.6 s.

Finally, the word "bay" contains the voiced stop /b/, the dipthong /eI/ which is a downward frequency sweep of the first formant from 600 Hz to 400 Hz and a slight upwards frequency sweep of the second formant from 2 kHz to 2.5 kHz. The first and second formants are clear in the plot. This speech segment lasts 0.6 s.

4.6 Long speech example

To show the capability of the system to process speech longer than one second, a 2.4 second example of the spoken numbers "four five six seven" was processed. The primary bottleneck in processing the data was in the transfer of data from the Vectra HP to its hard drive. Speeds of processing averaged about 2 minutes for one second of speech. This indicates an average disk transfer speed of about 1.3 Mbytes per minute, or 22.2 kbytes per second. While the length of the data makes it difficult to make out details on this plot, the high frequency fricatives in "six" are easily found.

The 120 seconds processing time per 1 second of speech is still very fast compared with 3,600 seconds of processing time for 1 second of speech mentioned in [2]. It should be noted that the latter timing refers to a more complex cochlear model than the one implemented by the VLSI cochlea model chip, although there is no reason why analog VLSI rectifiers, low pass filters, and lateral inhibition

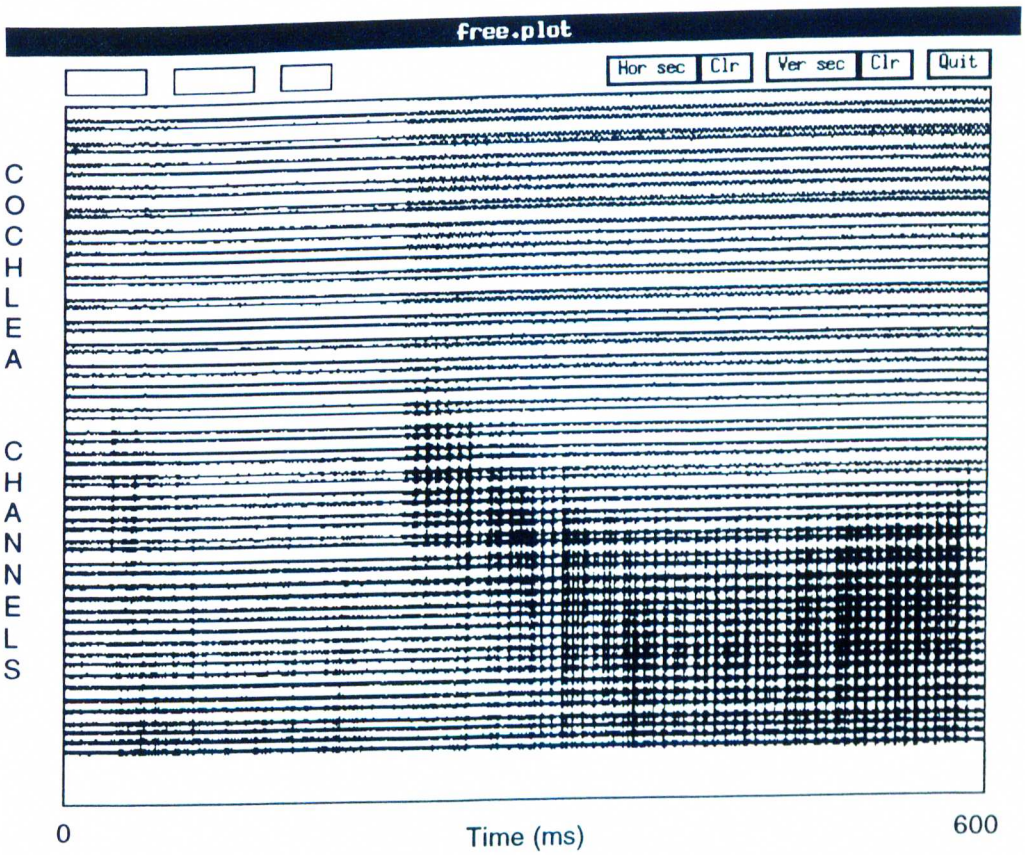

Figure 4.13: Response to the spoken word "free" (Total time 600ms)

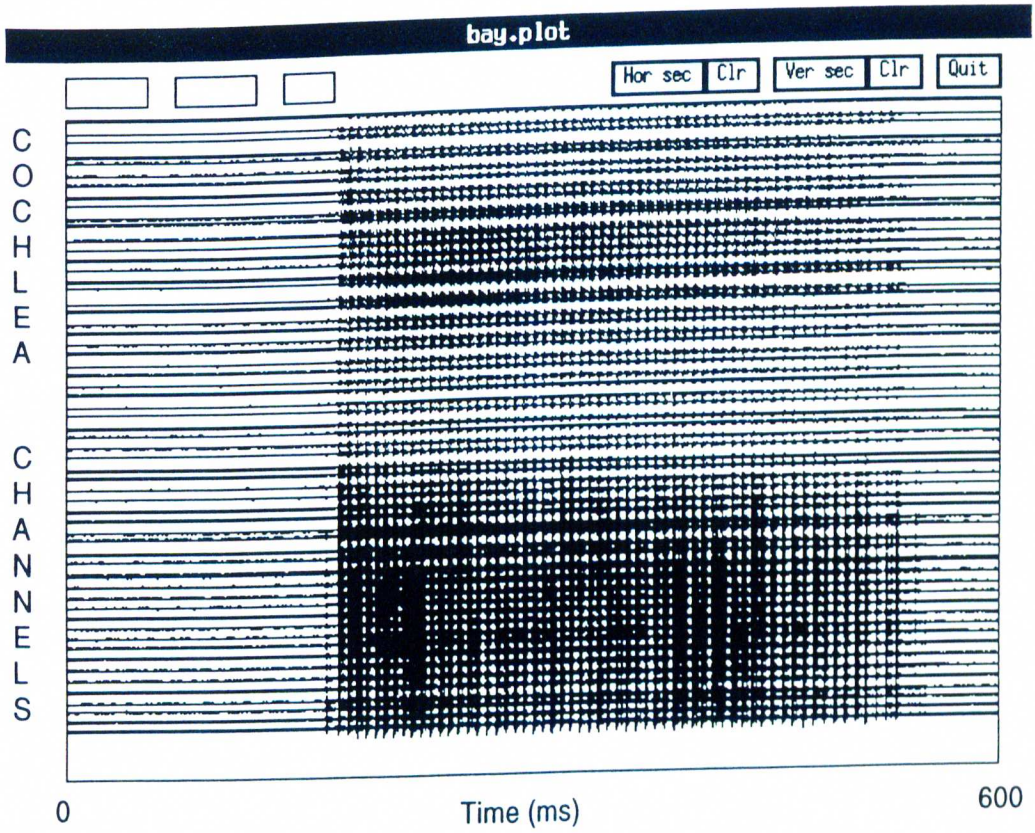

Figure 4.14: Response to the spoken word "bay" (Total time 600ms)

networks cannot be added on to the current basilar membrane model chip with little extra computational expense.

4.7 Discussion of Results

Judging from the results of examples of both short stationary audio, and longer non-stationary audio processed through the VLSI cochlea system, the system appears to be properly performing its role of cochlear processing. Many well-known speech characteristics are reflected in the processed data plots, and they are obtained at a speed much faster than computerized cochlea models.

A useful set of computer programs have been developed to collect, convert, and transfer data from the VLSI cochlea chip at high speeds. It is fairly easy to use, and integrated well into the ESPS signal processing system.

Speeds achieved with the VLSI cochlea model are far faster than similar computer models. Inherently, the VLSI model speed is nearly real-time, the only limit to its speed is the limitations of the data acquisition system. In this case, the primary bottleneck is the speed the hard disk can be written to.

It is unfortunate that the VLSI cochlear chip could be safely operated with only +/- 4V supplies. The op-amps of the SCF filters are the primary cause of dissipated power in the cochlea chips. With 104 op-amps on the chip, there is little room for power hungry op-amps. Future version of the VLSI cochlea chip will need to be re-engineered with op-amps that draw less current so that the supply voltages can be raised, thus increasing the dynamic range of the chip, and also providing easier interface with some logic families. Dynamic range is an important consideration for this kind of a bandpass filter, since each channel

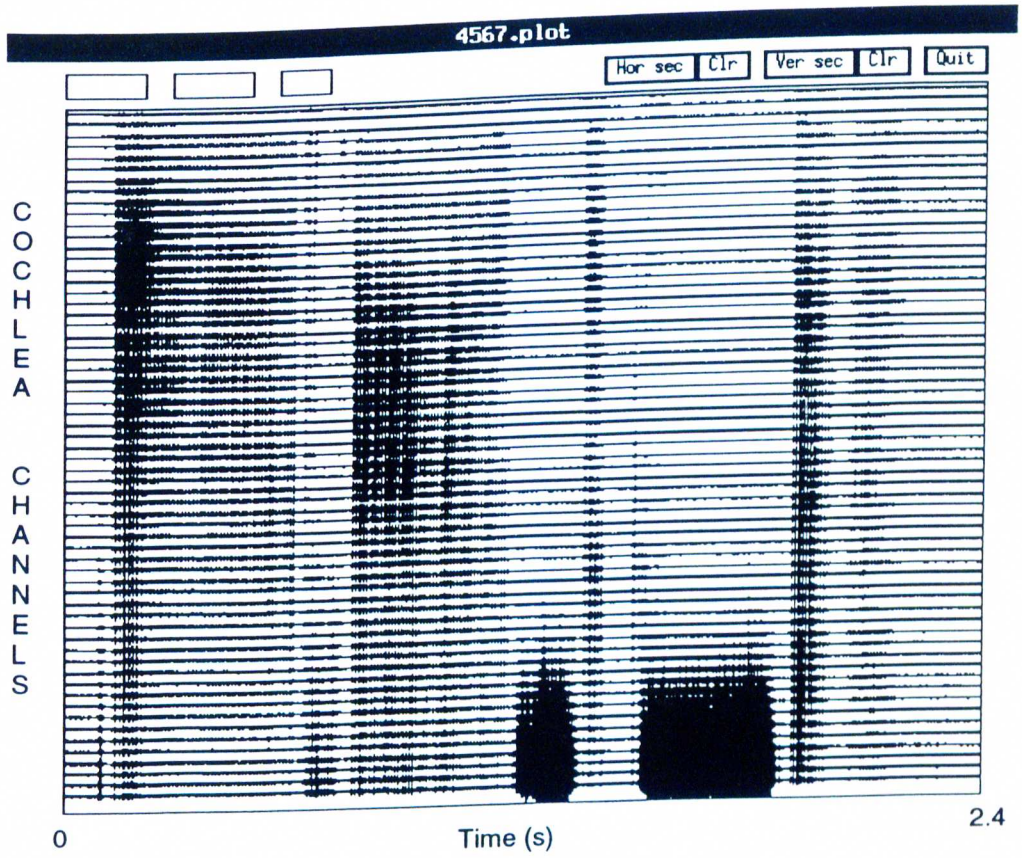

Figure 4.15: Response to spoken words "four five six seven" (Total time 2.4s)

outputs very little of the total input signal power, so it is preferable to have as large and input signal as possible to increase the output signal to noise ratio.

4.8 Future research

The current VLSI cochlea model has 32 channels per chip. While for SCF circuits, this is quite an achievement given the high order and the low frequency poles of the cochlea filters, there is clearly an advantage in trying to put even more filters on a single chip. Other technologies such as switched-current (SI) filters might allow this. SI filters utilize the capacitance of the gates of MOS FETs to store charge, and of course this change controls the current flowing through the device. SI suffers even more than SC from clock-feedthrough problems, and also has problems with transistor mismatch. There has already been some research into these two known problems [14] [15].

The VLSI cochlea model is only one part of a full cochleal processing system. As mentioned before, there are many other processing steps required to turn the "basilar membrane" representation of speech into a "cochlear nerve" representation. These rectifiers, low-pass filters, compressive non-linearities, and lateral inhibitory networks (LIN) could also be implemented in VLSI.

There is also the possibility of implementing a VLSI model of primary auditory cortex (AI), where more complex audio features are represented. In AI, single and multi-unit recordings have revealed a spatially ordered tonotopic axis, alternating bands of binaural response properties, which run perpendicular to isofrequency planes, and encoding of the shape of the acoustic spectrum according to spectral gradients across the isofrequency planes [16]. It is possible that

a VLSI model of some or all of these responses could be developed, using the responses of VLSI cochlea chip and other analog VLSI cochlea model extensions as input.

Finally, there is the possibility of using the VLSI cochlea model and/or a VLSI AI model as a front-end processor for a speech recognition system based on neural networks or hidden Markov models [17].

4.9 Conclusions

A set of hardware and software elements were designed in order to create a data acquisition interface of the switched-capacitor cochlea chip of Lin. The acquisition system included clocking mechanisms, A/D and D/A interfaces, channel bank select mechanisms, smoothing filters, and data acquisition software. The software was designed to allow for speech data to be efficiently processed by the cochlea model and recorded in a form which was importable to pre-existing signal analysis software. Several tests were performed to show the usefulness of the system, including single tone, double tone, vowel, isolated word, and connected word examples. The system performed appropriately, and is a fast way of performing cochlear processing of speech data.

Appendix A

Source Code for cochint.c

```
/* VSLI Cochlea Model System Interface Program */

/* system includes */
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <time.h>

/* FLASH12 includes */
#include "hsa.h"
#include "cio.h"
```

```
/* defines */
#define i_block 8192  /* input blocksize */
#define ESC 0x1b
#define BASE_ADDRESS 0x2300
#define sf short far

/* Global Variables */
void initialize_flash();
int slength;      /* sampling period in nanosecond */
long length;      /* length of audio file */
char title[80];  /* title of audio file */
short md=EGAHI,dvr=EGA; /* graphics variables */

int bank_codes[4]={239,253,247,251};  /* bank select codes */

/* Measurement codes */
char Measurements[] = { CHAN_0 | TIMES_20 | BIPOLAR | SNGL_POS ,
CHAN_1 | TIMES_20 | BIPOLAR | SNGL_POS ,
CHAN_2 | TIMES_20 | BIPOLAR | SNGL_POS ,
CHAN_3 | TIMES_20 | BIPOLAR | SNGL_POS ,
CHAN_4 | TIMES_20 | BIPOLAR | SNGL_POS ,
CHAN_5 | TIMES_20 | BIPOLAR | SNGL_POS ,
CHAN_6 | TIMES_20 | BIPOLAR | SNGL_POS ,
CHAN_7 | TIMES_20 | BIPOLAR | SNGL_POS ,
```

```
CHAN_8 | TIMES_20 | BIPOLAR | SNGL_NEG ,
CHAN_9 | TIMES_20 | BIPOLAR | SNGL_NEG ,
CHAN_10 | TIMES_20 | BIPOLAR | SNGL_NEG,
CHAN_11 | TIMES_20 | BIPOLAR | SNGL_NEG,
CHAN_12 | TIMES_20 | BIPOLAR | SNGL_NEG,
CHAN_13 | TIMES_20 | BIPOLAR | SNGL_NEG,
CHAN_14 | TIMES_20 | BIPOLAR | SNGL_NEG,
CHAN_15 | TIMES_20 | BIPOLAR | SNGL_NEG };

/* print centered text */
void center_print(char *s,int y)
{
 gotoxy(((80-strlen(s))/2),y);
 cprintf("%s",s);
}

/* create title page */
void title_page()
{
 clrscr();
 textbackground(1);
 textcolor(14);
 highvideo();
 center_print("VLSI Cochlea Model Interface System\n",3);
 textbackground(0);
```

```
 normvideo();
 center_print("Press any key to continue...",7);
 getch();
}



/* turn on D/A otput */

void run_anout()
{
 RunMode(RUN_NOBUF_REG);
 dac_ctrl(ARBITRARY_WAVEFORM | BIPOLAR_5V, ARBITRARY_WAVEFORM |
  BIPOLAR_5V);
 set_delay(slength,NANOSECOND);

 RunMode(RUN_BUF_REG);
}

/* read audio file */
void read_audio(char *fn,long offset,long r_length)
{
 int i,j;       /* general index */
 int fh;  /* file handle */
 int num_pointers;  /* number of pointers for audio data */
 unsigned int data,tmp; /* temporary data */
```

```
 static short buffer[8192]; /* buffer used to transfer data */
 long time,addr; /* time of data and address */

 printf("\nOpening file...\n");
 fh = _open(fn,O_RDONLY);
 lseek(fh,offset,SEEK_SET);
 printf("Reading in acoustic data..offset=%ld size=%ld\n...please wait.\n",
offset,r_length);
 system("hpmode speed high");
 time = 0x100000L;
 printf("Loading DAC RAM...");
 addr=0;
 for(i=0;i<16;++i)
{
for(j=0;j<8192;)
if((addr+8192)<r_length)
{
_read(fh,buffer,16384);
addr+=8192;
j+=8192;
}
else
if(addr<r_length)
{
_read(fh,&buffer,(2*(r_length-addr)));
```

```
j+=(r_length-addr);
addr+=(r_length-addr);
}
else
buffer[j++]=32767;

printf(".");
LoadDACBuffer((short far*)&buffer,8192,time);
if(time==0x100000L)
time=-1L;
}

  printf("\nWaveform loaded.\n");
  _close(fh);
  LoadDACBuffer((short far *)&buffer,0L,-2L);

  printf("\n%ld data points read in.\n\n",r_length);

  system("hpmode speed low");

  printf("press any key to continue...\n");

  getch();
}
```

```
/* initialize WBFLASH-12 card */
void initialize_flash()
{
 SetBase(BASE_ADDRESS);          /* set card base address */
 in_8536();                      /* Initialize Main 8536 */
 PointToDaughter(); /* Talk to Daughterboard */
 in_8536(); /* Initialize Daughter 8536 */
 PointToMainBoard(); /* point to main card */
 pll(256,256); /* set pll */
 DecompressEEPROM(); /* read in calibration data */

 BeSingleEnded();
}

/* print menu choices */
char print_choices()
{
 clrscr();
 printf("Enter Data Acquisition Choice:\n");
 printf("\n1...Demo Mode\n");
 printf("2...Process digitized data\n");
 printf("3...Digitize analog audio data\n");
 printf("4...Output digitzed analog data for examination\n");
 printf("0...Quit\n\n");
 return(getch());
```

```
}

/* perform demo mode */
void demo_mode()
{
 int i,k,ybase,dis_bank=0;
active_page, /* current page being written to */
visual_page, /* current displayed page */
holder, /* used to swap pages */
inch; /* input character */
 long sample_t=10000L; /* default sample period */
 unsigned int buff[640]={0}; /* A/D input buffer */

 SetDataDirection(BASE_ADDRESS,0x00);
 OutputDigitalByte(BASE_ADDRESS,bank_codes[0]);
 printf("press keys 0-3 to select 16-channel bank\n");
 printf("Type + or - to increase or decrease sample period\n");
 printf("Type q to quit\n\n");
 printf("press any key to begin demo mode...\n");
 getch();

 set_delay(sample_t,NANOSECOND);
 No_Trig(640L);

 initgraph((int far *)&dvr,(int far *)&md,"");
```

```
 LoadSRAM(16,(char far*)&Measurements,8192);
 wr_latch((short)Measurements);

 active_page=1;
 visual_page=0;
 setactivepage(active_page);
 setvisualpage(visual_page);

do
{
do
{

TakeTrace(16,HSCard->Chan_and_Range);
CollectTraceData(-0,640,(sf *)&buff,16);

for(i=0;i<640;++i)
 buff[i]=(buff[i]-32768) >>9;

cleardevice();

for(k=0;k<16;++k)
{
```

```
ybase=(16-k)*20;
moveto(0,ybase - buff[k]);
for(i=0;i<640; i+=16)
lineto(i,ybase - buff[i+k]);
}

setactivepage(active_page);
setvisualpage(visual_page);

holder=active_page;
active_page=visual_page;
visual_page=holder;

}while(!kbhit());
 inch=getche();
 if(inch=='0')
dis_bank=0;
 if(inch=='1')
   dis_bank=1;
 if(inch=='2')
   dis_bank=2;
 if(inch=='3')
   dis_bank=3;
 if(inch=='+')
   {
```

```
    sample_t *= 1.2;
    set_delay(sample_t,NANOSECOND);
   }
 if(inch =='-')
   {
    sample_t *= 0.833;
    set_delay(sample_t,NANOSECOND);
   }
 if(inch=='q')
   dis_bank=-1;
 SetDataDirection(BASE_ADDRESS,0x00);
 OutputDigitalByte(BASE_ADDRESS,bank_codes[dis_bank]);

}while(dis_bank != -1);

closegraph();
}

/*  acquire certain amount of A/D data */
void do_acquisition(long start, /* place to start D/A data */
    long stop,  /* place to end D/A data */
    int bank) /* channel bank to record from */
{
 long l_length; /* actual length to record from */
```

```
 SetDataDirection(BASE_ADDRESS,0x00);
 OutputDigitalByte(BASE_ADDRESS,bank_codes[bank]);

 l_length=(stop-start)*16L;

 No_Trig(l_length);

 LoadSRAM(16,(char far*)&Measurements,8192);
 wr_latch((short)Measurements);

 run_anout();
 TakeTrace(16,HSCard->Chan_and_Range);

}

/* acquire 1 channel of A/D data */
void do_acquisition_1(long l_length)
{

 SetTriggerSource(0,128);
 PreTrigger = 100;
 PostTrigger = l_length;
 TriggerTimeout = 1000000;
 TakeOneTrace(0,TIMES_4);
```

```
}

#define buff_size 1000
#define buff_len (buff_size * 16)

/* process a frame of data */
void process_sub(int fh,long ad_length)
{
FILE *fp;
int  i,j,k, /* loop variables */
num_passes, /* number of passes */
amount; /* how much data is written to disk */
long  addr, /* where in A/D buffer memory we are */
left, /* how much is left to write to disk */
int  bank; /* which bank we are reading from */
static unsigned int out_buff[buff_len]; /* disk output buffer */

printf("Processing Cochlear Response\n");

for(bank=0;bank<4;++bank)
{
printf("Bank #%d\n",bank);
do_acquisition(0,ad_length+overlap,bank);

system("hpmode speed high");
```

```
num_passes=(ad_length/buff_size)+1;
printf("Gathering data in %d passes...\n",num_passes);
addr=0L;
left=ad_length*16L;
printf("Writing to file...");

CollectTraceData(addr,overlap,(sf *)&out_buff,16);
addr=overlap;

for(i=0;i<num_passes;++i)
{
printf(" . ");
left=left-buff_len;
if(left>0)
amount=buff_len;
else
amount=buff_len+left;
printf("amount=%d addr=%ld...",amount,addr);
CollectTraceData(addr,amount,(sf *)&out_buff,16);
for(j=0;j<buff_size;j=j+1)
if((j*16)<amount)
_write(fh,out_buff+(j*16),32);
addr=addr+buff_len;
}
system("hpmode speed low");
```

```
}

printf("Done!\n");

printf("press any key\n");
getch();
}

/* record a channel of A/D data for later use */
void record_data()
{
 FILE *fp;  /* file pointer (ASCII stream)*/
 int fn; /* file handle (binary file)*/
 char   file_name[80], /* base file name */
t[75], /* keyboard input buffer */
header_file[80],/* header file name */
data_file[80]; /* data file name */
 char *title; /* title of data */

 int i,j,k,num_passes,amount;
 long addr,ad_length,left;

 unsigned int out_buff[buff_size];

 printf("\nEnter sampling period in ns: ");
```

```
scanf("%d",&slength);

printf("\nEnter Length of data (points): ");
scanf("%ld",&length);

printf("Length=%ld\n",length);

printf("\n\nEnter file name: ");
scanf("%s",file_name);

printf("\nEnter Title: ");
t[0]=75;
title=cgets(t);

printf("\nWriting file...please wait...\n");

strcpy(header_file,file_name);
strcat(header_file,".hed");

strcpy(data_file,file_name);
strcat(data_file,".dat");

printf("Header file=%s Data file=%s\n",header_file,data_file);

fp=fopen(header_file,"w");
```

```
 fputs(title,fp);
 fprintf(fp,"\n%ld %d\n",length,slength);
 fclose(fp);

 if((fn=_creat(data_file,0))==-1)
{
printf("Open failed\n");
exit(-1);
}

 slength=slength*2;

 set_delay(slength,NANOSECOND);

 printf("Ready to Record, Press any key to begin:\n");

 getch();
 do_acquisition_1(length);

 num_passes=(length/buff_size)+1;

 printf("Gathering data in %d passes...\n",num_passes);

 addr=0;
```

```
 left=length;

 printf("Writing to file...");

 for(i=0;i<num_passes;++i)
{
printf(" . ");
left=left-buff_size;
if(left>0)
 amount=buff_size;
else
 amount=buff_size+left;
printf("amount=%d addr=%ld\n",amount,addr);
CollectTraceData(addr,amount,(sf *)&out_buff,1);
_write(fn,out_buff,(2*amount));
addr=addr+amount;
}

 printf("Done! Press any key\n");
 _close(fn);
 getch();
}

/* process full length data */
void full_process()
```

```
{
char data_file[80];  /* speech data file name */
char file_name[80];  /* file base name */
char header_file[80];  /* head file name */
FILE *fp;      /* file stream pointer */
long length_left;  /* file length left to process */
long overlap;
#define frame_len 15000L/* length of processing frame */
clock_t start_time; /* timer variable */
long offset=0L; /* offset variable */
int fh; /* binary file handle */

{
printf("\nEnter digitized audio filename: ");
scanf("%s",file_name);

strcpy(data_file,file_name);
strcat(data_file,".dat");
strcpy(header_file,file_name);
strcat(header_file,".hed");

printf("Header file=%s Data file=%s\n",header_file,data_file);

if((access(data_file,0)+access(header_file,0)))
{
```

```
 printf("There is no such file!\n");
 sound(600);
 delay(250);
 nosound();
 }

} while(((access(data_file,0))==-1) || ((access(header_file,0))==-1));

 fp = fopen(header_file,"r");

 fgets(title,40,fp);
 fscanf(fp,"%ld %d",&length,&slength);

 highvideo();
 printf("\nTitle: %s\n",title);
 normvideo();
 printf("Length: %ld\n",length);
 printf("Sampling Period: %d ns\n",slength);
 overlap=5000.0/(slength/1000.0)                    /* shoot for 5ms overlap */
 fclose(fp);

 printf("\nOffset for processing: ");scanf("%ld",&offset);
 printf("Length of processing: ");scanf("%ld",&length);

 offset=offset*2;   /* convert to bytes */
```

```
 length=length/8;   /* convert to A/D samples */

 set_delay(slength,NANOSECOND);

 system("del audio.out");

 if((fh=_creat("audio.out",0))==-1)
{
printf("Open failed\n");
exit(-1);
}

 start_time=clock();

 length_left=length;

 do
 {
        if(length_left > frame_len)
                {
printf("Running full frame...\n");
initialize_flash();
printf("calling read_audio offset=%ld size=%ld\n",
offset,(frame_len*8));
```

```
read_audio(data_file,offset,((frame_len+overlap)*8));
offset+=(frame_len*8*2);
length_left=length_left-frame_len;
process_sub(fh,frame_len,overlap);
                }
                else
                {
printf("Running partial frame...\n");
                initialize_flash();
printf("calling read_audio offset=%ld size=%ld\n",
offset,(length_left*8));
read_audio(data_file,offset,((overlap+length_left)*8));
 process_sub(fh,length_left,overlap);
                length_left=0;
                }
        } while(length_left != 0);

 _close(fh);
 printf("Ellapsed time = %ld\n",clock()-start_time);
 printf("Done!\nPress any key to continue...");
 getch();


}
```

```
/* preview speech */
void preview()
{
char  title[40],    /* speech file title */
file_name[80], /* base file name */
data_file[80], /* data file name */
header_file[80];/* header file name */
FILE  *fp; /* ASCII file stream */
long  offset, /* data offset */
listen_length;  /* length of data to listen to */

{
printf("\nEnter digitized audio filename: ");
scanf("%s",file_name);

strcpy(data_file,file_name);
strcat(data_file,".dat");
strcpy(header_file,file_name);
strcat(header_file,".hed");

printf("Header file=%s Data file=%s\n",header_file,data_file);

if((access(data_file,0)+access(header_file,0)))
{
 printf("There is no such file!\n");
```

```
 sound(600);
 delay(250);
 nosound();
 }

} while(((access(data_file,0))==-1) || ((access(header_file,0))==-1));

 fp = fopen(header_file,"r");

 fgets(title,40,fp);
 fscanf(fp,"%ld %d",&length,&slength);

 highvideo();
 printf("\nTitle: %s\n",title);
 normvideo();
 printf("Length: %ld\n",length);
 printf("Sampling Period: %d ns\n",slength);
 fclose(fp);

 length=length/8;
 set_delay(slength,NANOSECOND);

 printf("Enter offset: ");scanf("%ld",&offset);
 printf("Enter listening length: ");scanf("%ld",&listen_length);
```

```
initialize_flash();
read_audio(data_file,(offset*2),listen_length);
 run_anout();
 printf("\nWaveform is now being generated for review.\n");
 printf("Press any key to continue...");
 getch();

 RunMode(STOP_MODE);
}

/* main program */
void main()
{
 char choice; /* menu choice */
 int  quit=0; /* quit flag */
 int  j;

 title_page();
 initialize_flash();

 do
 {
  choice=print_choices();
  switch(choice)
 {
```

```
case '1': demo_mode();
  break;
case '2': full_process();
  break;
case '3': record_data();
  break;
case '4': preview();
  break;
case '0': quit=1;
  break;
default : break;
}
  } while(quit == 0);

}
```

Bibliography

[1] K. Wang and S. Shamma. Zero-crossings and noise suppression in auditory wavelet transformations. Technical Report 94, University of Maryland Systems Research Center, 1992.

[2] M. J. Farmelo. An implementation of a cochlear model for speech processing. Master's thesis, University of Maryland, 1988.

[3] J. Lin. *Mixed-Mode VLSI Implementations of Auditory Wavelet Transforms Using Switched-Capaitor Circuits.* PhD thesis, University of Maryland, 1993.

[4] R. F. Lyon and C. Mead. An analog electronic cochlea. *IEEE transactions on Acoustics, Speech, and Signal Processing*, 36(7), July 1988.

[5] L. Watts, D. A. Kerns, and R. F. Lyon C. Mead. Improved implementation of the silicon cochlea. *IEEE Journal of Solid-State Circuits*, 27(5):692–700, May 1992.

[6] W. Liu, A. G. Andreou, and M. H. Goldstein Jr. Analog vlsi implementation of an auditory periphery model. In *Conference on Information Science and Systems*, 1991.

[7] K. Nagaraj. A parasitic-insensitive area-efficient approach to realizing very large time constants in switched capacitor circuits. *IEEE Transactions on Circuits and Systems*, 36(9):1210–1216, September 1989.

[8] J. Lin, W-H. Ki, K. E. Thompson, and S. Shamma. Cochlear filters design using a parallel dilating- biquad switched-capacitor filter bank. In *IEEE Intenrational Symposium on Circuits and Systems*, pages 2053–2056, May 1992.

[9] R. Gregorian and G. Temes. *Analog MOS Integrated Circuits*. John Wiley & Sons, 1986.

[10] A. Oppenheim and R. Shafer. *Digital Signal Processing*. Prentice-Hall, Inc., 1975.

[11] J. Shore and A. Parker. *Entropic Signal Processing System (ESPS 3) Technical Memoranda and Applications Notes*. Entropic Speech, Inc., 1989.

[12] Entropic Speech, Inc. *waves+ User's Manual*, 1990.

[13] L. Rabiner and R. Shafer. *Digital Processing of Speech Signals*. Prentice-Hall, Inc., 1978.

[14] H. C. Yang, T. S. Fiez, and D. J. Allstot. Current-feedthrough effects and cancellation techniques in switched-current circuits. In *IEEE International Symposium on Circuits and Systems*, pages 3186–3188, May 1990.

[15] D. Vallancourt, Y. P. Tsividis, and S. J. Daubert. Current copier cells. *Electronics Letters*, 25:644–646, May 1989.

[16] S. Shamma, J. Fleshman, P. Wiser, and H. Versnel. Organization of response areas of ferret primary auditory cortex. *Journal of Neurophysiology*, 69:367–383, Feb. 1993.

[17] W. Byrne, J. Robinson, and S. Shamma. The auditory processing and recognition of speech. In *DARPA Speech and Natural Language Workshop*, Oct. 1989.