1

THE MICROSCOPE
MICROSCOPE
A BENTON & BOWLES COMPANY

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.

A Novel Speech Spectrograph System

by

William Burchill

B. Eng.

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

Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University

June 1985

copyright

1985, William Burchill

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 0-315-39417-X

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

A Novel Speech Spectrograph System
submitted by William Burchill, in partial fulfilment of the
requirements for the degree of Master of Engineering.

Thesis Supervisor

8/21/85

Chairman,

Department of Systems

and Computer Engineering

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

June, 1985

Abstract

The study of acoustic phonetics has been revolutionized by the invention of the speech spectrograph. It was the introduction of this device in the 1940's that made possible extensive work in this field. However, despite its significance the spectrograph has undergone remarkably little development from the original analog designs.

This thesis describes the design of a novel spectrographic system based on DSP techniques. It begins with a basic examination of acoustic phonetics. A unified and self-consistent development of relations between the Z transform, filter banks, the DFT and a complex scanning filter is presented. An in-depth examination of optimizing an FFT algorithm to a specific processor architecture provides a focus.

Acknowledgement

I would like to thank my thesis advisor Dr. H. M. Hafez and the personnel of the Advanced Processing Systems division, D.I.T., of the Communications Research Center of Canada, for making this work possible.

March 28/85

Table of Contents

	Page
1 Introduction	
1.1 Introduction	1
1.2 Scope and thesis organization	3
2 Fundamental Phonetics	
2.1 Introduction	5
2.2 Articulatory phonetics	5
2.3 English consonants	8
2.4 English vowels	12
2.5 Coarticulation and emphasis	15
3 Speech spectrographs	
3.1 Introduction	17
3.2 Spectrograph production	18
3.3 Interpretation of spectrographs	21
3.4 Applications	28
4 The proposed speech spectrograph system	
4.1 Introduction	30
4.2 Definition of spectrum and analysis structures	31
4.3 Analysis parameters for a speech spectrograph	44

5 Coding the FFT	Page
5.1 Introduction	53
5.2 The flow diagram	55
5.2.1 Deriving the flow diagram	55
5.2.2 Observations	62
5.3 Architectural considerations	66
5.3.1 An introduction to the TMS32010 architecture	66
5.3.2 Implications	69
5.4 The basic DIF0-R4B	77
5.5 The DIF2-R4B	92
5.6 The DIF3-R4B and the DIF4-R4B	96
5.7 Ordering the input sequence	99
5.8 The doubling algorithm	103
5.9 Benchmark comparison	106
6 Overview of the speech spectrograph	
6.1 Introduction	108
6.2 Basic configuration	109
6.3 Task allocation	111
6.4 DSP software	111
6.4.1 Input	113
6.4.2 Hanning Window	114
6.4.3 FFT and doubling algorithm	114
6.4.4 Magnitude estimation	115

6.4.5 Dynamic range compression	116
6.4.6 Interpolation	117
6.4.7 Output	118
6.5 DSP hardware module	118
6.6 Alternate configurations	124
7 Conclusion	
7.1 Introduction	126
7.2 Current status	127
7.3 Comparison to Kay Sonogram	128
7.4 Concluding remarks	130
References	133
Appendix	
A.1 Indirect addressing on the TMS32010	137
A.2 Code sizes and timings	140

List of Figures

Name	Page
Fig. 2.1 The Human vocal tract	6
Fig. 3.1 An early spectrograph device	19
Fig. 3.2 Vowels from: Heed, Hid, Head, Had.	22
Fig. 3.3 Vowels from: Hod, Hood, Food.	22
Fig. 3.4 The dipthong 'I'.	24
Fig. 3.5 The phrase, 'She came back'.	24
Fig. 3.6 Famous phrase, 'We were away a year ago.'	27
Fig. 4.1 Two possible regions for calculating a signal spectrum.	32
Fig. 4.2 A FIR structure for sliding spectrum analysis at z_k .	35
Fig. 4.3 A recursive structure for spectrum analysis at z_k .	37
Fig. 4.4 A recursive structure for the DFT at $\exp(j2\pi k/N)$, $k=0,1,2,\dots,N-1$.	39
Fig. 4.5 A bank for the DFT at $\exp(j2\pi k/N)$.	39
Fig. 4.6 DFT filter bank response with $N=8$.	45
Fig. 4.7 Effect of a Hanning window on DFT frequency response.	45
Fig. 4.8 Complex spanning filter view of the DFT.	46

	Page
Fig. 4.9 Spectrum illustrations for Fig. 4.8.	46
Fig. 5.1 FFT flow diagram.	63
Fig. 5.2 4 point DFT's.	65
Fig. 5.3 Block diagram of the TMS32010.	67
Fig. 5.4 Comparing the DIT and DIF viewpoints.	75
Fig. 5.5 DIF-R4B examples.	76
Fig. 5.6 Flow diagram for R4B grouped equations.	80
Fig. 5.7 A flow diagram for the basic DIF0-R4B.	82
Fig. 5.8 A 34 instruction DIF0-R4B.	90
Fig. 5.9 The basic 33 instruction DIF0-R4B.	91
Fig. 5.10 The DIF2-R4B.	97
Fig. 5.11 Benchmark comparison	107
Fig. 6.1 A novel speech spectrograph system.	110
Fig. 6.2 DSP software for the speech spectrograph.	112
Fig. 6.3 Analog interface.	121
Fig. 6.4 DSP module card.	122
Fig. 7.1 Comparison to Kay Sonogram	129

Chapter 1

Introduction

1.1 Introduction

The study of acoustic phonetics has been revolutionized by the invention of the speech spectrograph. It was the introduction of this device in the 1940's that made possible extensive work in this field. However, despite its significance, the spectrograph has undergone remarkably little development from the original analog designs. Published accounts of digital devices indicate serious shortcomings with each. As a result, the current laboratory standard remains the Kay Sonogram, a primarily analog device of considerable expense.

In many modern research facilities speech is routinely processed or even synthesized by entirely digital means. This has led to the perceived need for a digitally based spectrograph which would readily lend itself to integration with a larger speech research system. With such a device, speech files could be subjected to alternate processes and their spectral effects evaluated. If the device were also very fast an interactive mode

would be possible.

The past few years have seen the proliferation of microcomputers into every environment. More recently, the DSP (digital signal processing) field has received a shot in the arm by the introduction of specially optimized high performance/cost ratio DSP microprocessors.

It should be possible to design a DSP module in the form of a standard interface card to be added to an existing microcomputer. With appropriate software the combination would be the needed spectrograph device and at the same time retain all other microcomputer functions. It is the author's assessment that the incremental cost of the addition should be less than the current standard device by a sum in excess of one order of magnitude.

To appreciate what is required of a spectrograph it is necessary to know something of articulatory phonetics and its relation to the spectral composition of speech sounds. The high level design starts with a block diagram specification and the partitioning of tasks to hardware and software. At the lowest level there is the specification of electronic circuitry and the coding of software. Along the way, microcomputer architecture and digital signal processing techniques play important roles. At the

implementation phase, which has also been part of this work, a familiarity with several development systems is essential.

In summary, the spectrograph design is inherently a systems design in that it is multileveled and draws on several disciplines.

1.2 Scope and thesis organization

Chapter 2 begins with a basic examination of acoustic phonetics. English language sounds are categorized by their places and manners of articulation. Correlations between these and a spectrographic representation are presented in chapter 3 with applications. The purpose and utility of a speech spectrograph is explained.

Chapter 4 presents an introduction to the principles of digital spectral analysis. A unified and self-consistent development of relations between the Z transform, filter banks, the DFT and a complex scanning filter is presented. Considerations of complexity are used with the analysis requirements inferred from chapters 1 and 2 to establish the parameters and the analyzing structure of an effective

spectrographic device.

There are many aspects to the design and development of a complete speech spectrograph system. Chapter 5 provides a focus by presenting an in-depth examination of optimizing an FFT algorithm to a specific processor architecture. The design methodology and its results are presented.

Chapter 6 presents an overview of the speech spectrograph, discussing briefly the DSP software and hardware which form an integrated system. In chapter 7 the current status of the work is presented with plans for future development. Some conclusions are drawn about the results of this work and spectrographic analysis in general.

Chapter 2

Fundamental Phonetics

2.1 Introduction

An understanding of human speech sounds is necessary to appreciate the criteria by which they are distinguished.

Beginning with the origins of speech sounds, section 2.2 of this chapter presents some knowledge of articulatory phonetics [1]. The physical structure of the vocal tract is described followed by several methods of articulating sounds. In sections 2.3 and 2.4 means of articulating single consonants and vowels of the English language are presented. Section 2.5 considers the effects of coarticulation and emphasis.

2.2 Articulatory phonetics

In Fig. 2.1 the human vocal apparatus is pictured. It begins at the bottom of the figure with the vocal cords of the larynx. Not depicted are the lungs which for the production of nearly all

Fig. 2.1 The Human vocal tract [1].

sounds force air up the trachea and through the larynx. The vocal cords are a pair of muscular flaps that may be contracted such that only a narrow slit shaped opening remains between them. This obstructs air flow and if the lungs force air through, the vocal cords are set vibrating producing a voiced sound. An unvoiced sound is produced if the vocal cords are relaxed. In this case the passage of air is fairly unobstructed and an unvoiced sound results. All vowels are voiced sounds. Consider for example the vowel sound of e as in the word 'heed'. If one's fingertips are placed on the outside of the throat opposite the larynx, the strong vocalic vibrations will be felt when the word 'heed' is said aloud. For contrast, if the experiment is repeated while saying an ssss sound like the first consonant of the word 'snake', the vibration is not felt. The latter test is an unvoiced sound.

Above the larynx is the pharynx. There are two routes for the air flow. One is the oral tract, the combination of the pharynx and the mouth. The other is the nasal tract, the passage behind the uvula and through the nasal cavity. Passage of air flow through the nasal tract is controlled by the velum, a soft muscular area in the upper rear of the mouth cavity. When contracted the velum raises the uvula sealing off the passage. This is not felt, but it is readily heard. Listen to the beginning sounds of the words 'be' and 'me'. The use of the lips, tongue

and larynx is identical in forming the two words. It is the addition of the nasal tract in the latter case that converts the b sound to an m sound. Sounds which use the nasal tract like m are therefore called nasals.

Considering the structure of the mouth it is seen that there are many possibilities for articulating sounds. These may be catagorized as to the place of articulation and the manner of articulation. There is more than one manner of articulating at each place.

To allow a specific description of the places of articulation the surface of the tongue is divided into six regions. These are the tip, blade, front, center, back and root as shown in Fig. 2.1. Also, the upper palate is divided into the alveolar ridge, the hard palate and the velum. The lips and teeth complete the list of articulatory surfaces. The positions of these surfaces should be clear from Fig. 2.1. A discussion of their use in making English consonants and vowels follows.

2.3 English consonants

In English there are eight basic places of articulation, that

is to say, eight basic places where air flow is obstructed in some manner.

There are four categories for the manner of articulation. There is the stop, fricative, approximant and the lateral.

The stop requires complete closure of the articulators making the sound. For example, 'be' and 'me' both begin by closing the lips to stop air flow through the mouth completely. Stops may be further categorized as oral and nasal in which case the beginning of 'be' belongs to the former and 'me' to the latter category.

In fricatives turbulent airflow is produced by forcing air between closely spaced articulators. The th sound of 'thee' and the s sound of 'see' are examples.

In an approximant the spacing is less close than for fricatives and so the air flow is not turbulent. The w of 'wield' and the y sound in 'yield' are examples.

The fourth manner, the lateral, has only one example in English. It is the l sound in 'lip'. This manner is identified by the contact of the roof of the mouth by the end of the tongue,

and one or both sides.

Previously mentioned were the eight basic places of articulation. With the manners of articulation now defined, the manner of articulating each consonant may be included with its discussion of place:

Bilabial - meaning the two lips are used. The three word progression 'pea, bee, me' makes an interesting example. The p sound is an unvoiced bilabial oral stop. The b sound is a voiced bilabial oral stop, and the 'm' sound is a voiced bilabial nasal stop.

Labiodental - meaning the lower lip and upper front teeth. The f and v sounds as in 'facts' and 'VAX' are unvoiced and voiced fricative examples respectively.

Dental - meaning the tongue tip or blade and the upper front teeth. The th sound of 'three' is an unvoiced fricative example.

Alveolar - meaning tongue tip or blade and the alveolar ridge. T in 'toe' is an unvoiced oral stop and s in 'sew' is an unvoiced fricative.

Retroflex - meaning the tip of the tongue and the back of the alveolar ridge. The r in 'row' can be a voiced approximant retroflex sound. It is not necessarily so. Another way to make an r sound is to pull the front of the tongue backwards such that it bunches up in the rear of the mouth.

Palato-Alveolar - meaning the blade of the tongue and the back of the alveolar ridge. The sh sound in 'she' is an unvoiced fricative.

Palatal - meaning the front of the tongue and the hard palate. The h sound in 'huge' is a voiced fricative.

Velar - meaning the back of the tongue and the soft palate. The c in 'cool' is an unvoiced oral stop.

The remaining consonants of English will not be described explicitly in order to limit the length of the discussion, but at this point sufficient categorizations have been presented to uniquely specify any English consonant. In summary only four factors are needed to effect the specification. These are:

- 1) Voiced or unvoiced
- 2) Oral or nasal
- 3) place of articulation (one of eight)
- 4) manner of articulation (one of four)

As an example, the th sound in 'the' is;

- 1) voiced
- 2) oral
- 3) dental
- 4) fricative

2.4 English Vowels

Vowel sounds may be distinguished from consonants in that they are always produced by a voiced and fairly unobstructed air flow. It is unfortunately very much more difficult to describe vowel sounds in terms of their articulation than it is to describe consonants. With vowels there are no distinct contact surfaces in use and there is at least theoretically a continuum of vowel sounds.

In practice a continuum is not used since this would lead to too many perceptual errors, but still the variation in the pronouncing of vowel sounds between individuals is much greater than for consonants. This is especially true if different dialects such as British and Canadian or even Ottawa center and Ottawa valley English are compared.

To make the best of a difficult situation what can be done is to describe a few distinct vowels to be used as reference points. Other vowels may then be described as intermediate in quality and hopefully articulation between an appropriate pair of reference vowels.

Consider the vowels in the sequence [l] 'heed, hid, head', and 'had'. In each of these vowels the tongue is moved to the front of the mouth and the height of the front of the tongue is greatest for ee in 'heed' and least for a in 'had'. These are called front vowels. The ee of 'heed' is a high front vowel. The i of 'hid' and the ea sound of 'head' are mid high and mid low front vowels respectively. In most people the mouth will be opened slightly wider for each vowel in the progression.

In three other reference vowels the highest part of the tongue is in the back of the mouth. The vowels in 'food', 'good', and the

a in 'father' are high, mid and low back vowels respectively.

Some English vowels are deceptive in that they are actually diphthongs, a sequence of two vowel sounds connected by a continual glide. The i sound of the word 'I' is an example. When said slowly, a brief ah sound like the a in 'father' can be heard which changes to an i sound like in 'it'. If the vowel pair ah - i is said at first slowly and then more quickly, the sound will be heard to approach 'I'. This assumes central Canadian pronunciation. As an example of a regional variation, in the southern United States the diphthong I is often shortened to a monophthong (one vowel sound). Instead of 'I thank you kindly, sir.', one hears, 'Ah thank you kahndly, suh.'

Describing vowel sounds by referencing the highest point of the tongue is a first approximation. Each vowel sound is actually dependent on the entire shape of the vocal tract [2]. This would include the location of each surface of the tongue and the lips.

In pronouncing 'heed' the lips are drawn open against the teeth whereas to pronounce the vowel in 'food' they are constricted into a short tube.

2.5 Coarticulation and emphasis

The preceeding sections on English consonants and vowels considered their individual articulation. When the individual sounds are combined with others as in forming syllables their articulations become modified. This is coarticulation. The effect can be felt when pronouncing the words 'geese' and 'gauze'. The articulation of the g in the former case is forward of the latter. In the word 'geese' the presence of a front vowel immediately after the g is anticipated by the forward articulation of the consonant. When the g is completed the tongue is close to the required position for the front vowel.

Coarticulation is not restricted to consonant - vowel combinations and also occurs between words, phrases, and sentences. It is a direct result of the inertia of the human vocal apparatus.

Sounds in combination are given different emphasis to convey changes in meaning. Volume changes to indicate important words or stressed syllables affect the intensity of the articulation. When speaking quickly vowels are frequently shortened or dropped altogether, and changes in the pitch pattern of a sentence can convert a statement to a question.

The many modifications of the basic English sounds in combination are not generally explicitly recognized by the untrained ear, yet few difficulties arise. However, these modifications make the recognition of speech sounds considerably more challenging when less natural means are employed.

Chapter 3

Speech Spectrographs

3.1 Introduction

A speech spectrograph is a visible presentation of speech sounds. The most common format is to present the relations between time, frequency and energy as a variable intensity plot on a two dimensional surface. The plot has a horizontal and a vertical axis representing time and frequency respectively. The energy at a given frequency - time point is represented by the intensity of the plot at that point. Generally linear scales are used for the time and frequency axis, but some form of dynamic range compression is usually employed in mapping intensities. Preemphasis of higher frequencies is also used to render these more visible.

The useful information apparent in high quality spectrographs is sufficient to permit their reading by trained observers.

Section 3.2 reviews some methods of producing speech spectrographs followed in section 3.3 by a brief discussion on

their interpretation. Section 3.4 presents some applications.

3.2 Spectrograph production

Speech spectrographic equipment has existed since the 1940's. The early devices were entirely analog and utilized some ingenious techniques [3], [4]. A simplified schematic of an original spectrograph is shown in Fig. 3.1. The device worked as follows: A recording of the speech to be analyzed was made on the tape loop. The tape loop was then played back sending the speech signal through the variable bandpass filter, initially set around 50 hz. Concurrently, a drum wrapped in electrically sensitive paper began to rotate, its rotation was mechanically linked to the tape player. The bandpass filter fed the electric stylus which in turn contacted the sensitive paper. Thus a line was drawn on the paper of intensity proportional to the amplitude of the filter's signal, which in turn was related to the speech energy within its band. An analysis of speech energy in the 50 hz band as a function of time was completed.

A mechanical link also existed between the stylus and the center frequency adjustment of the variable filter. By repeating the analysis with many different center frequencies and stylus

Fig. 3.1 An early spectrograph device [3].

offsets, the entire speech band was analyzed.

Typically the apparatus described would cover the frequency range from 70 to 3500 cycles in 200 steps, and the analyzing filter bandwidth was 300 hz. Analysis of a 2.4 second speech sample required only about 2.5 minutes [4].

In light of today's technology, several alternative procedures are apparent, the most flexible involving digital techniques.

Since the publication of the Cooley - Tukey algorithm for the FFT [5] made digital spectral analysis practical, several papers [6], [7], [8] and [9] have been published with methods for computer generation of speech spectrographs. However, the current laboratory standard spectrograph remains an expensive analog device.

In the Kay Sonogram the analysis procedure is a variation of that already described. The analyzing filter is in this case fixed and centered at a low intermediate frequency. The baseband speech signal is mixed with a frequency ramp which causes it to be swept past the analyzing filter, performing the analysis at I.F. The resulting signal is compressed logarithmically in dynamic

range by another analog circuit and digitized for storage. The stored signal is used to intensity modulate a monochrome video signal which in turn drives a raster scan display. The result is a very high quality spectrograph which serves as the standard by which others are evaluated.

3.3 Interpretation of spectrographs

Spectrographs of the seven reference vowels discussed in Chapter 2 appear in Fig. 3.2 and Fig 3.3. For each vowel, at least three bright horizontal bands called formants can be seen. These formants are concentrations of speech energy at the resonant frequencies of the vocal tract, and are characteristic of vowels and certain vowel - like sounds.

The lower three formants are called formant 1, formant 2 and formant 3 respectively, beginning with the bottom band.

There are also vertical striations visible throughout the formants. The period of these striations is the period of the vocal cord wave and their absence indicates an unvoiced sound.

Some interesting patterns are visible in Figs. 3.2 and 3.3.

Fig. 3.2 Vowels from, 'Heed, Hid, Head, Had'.

Fig. 3.3 Vowels from, 'Hod, Hood, Food'.

Figure 3.2 is the sequence of the four front vowels from the highest, ee in 'heed', to the lowest, a in 'had'. The heights of the second and third formants follow this pattern. This is true as well for the back vowels of Fig. 3.3, but the spacing between formant 2 and formant 3 is much greater in the latter case. Thus the front vowels may be distinguished from the back vowels by the spacing between formant 2 and formant 3, and the particular vowel within each category may be identified by the spacing between formant 1 and formant 2.

There is also a slight upward movement of formant 1 throughout the front vowels and a slight downward movement throughout the back vowels, but this is a less useful fact for identifying the vowels. The significant clues to the identity of a vowel are the relative positions of its formants.

The classification of vowels by a coarse description of their articulation was earlier described as a first approximation. However it is interesting to see that the sequence of articulatory positions from high to low, as well as the groups front and back are visible on speech spectrographs.

As an example of a dipthong the word 'I' is shown in Fig. 3.4. It can be seen that the formants begin at the positions of a vowel

'ah'

'I'

'i'

Page 24

Fig. 3.4 The dipthong, 'I'.

1	2	3	4	5	6	7	8

Fig. 3.5 'She came back'.

close to the a in 'father' and move continually to the positions of i in 'hid'.

The recognition of consonants is more difficult due to the many factors involved. In particular, stop consonants have little or no sound of their own. They must be recognized by their effect on neighbouring sounds and a knowledge of possible English words. Some effects are not so difficult to identify. Bilabial consonants tend to make the second and third formants of a neighbouring vowel droop slightly and velar consonants bring the second and third formants to a common point. Fricatives are identified by a wide bandwidth random noise pattern and voiced sounds by vertical striations.

Figure 3.5 shows the phrase, 'She came back'. Eight points of interest have been marked on the spectrograph to aid discussion. The first sound in the phrase is the voiceless fricative sh of 'she'. This is marked point (1) and is seen to be a burst of random noise. Point (2) is the vowel e of 'she'. It is recognized by the presence of strong voiced formants. The other vowels, a in 'came' and a in 'back' are similarly marked (4) and (7). They also show the characteristic strong voiced formants of vowels.

Point (3) is the c of 'came', an unvoiced velar stop consonant. The duration of this sound is less than the consonant at (1) and so the width of the noise burst is less. The second and third formants approach each other slightly because the consonant is velar, although this is not very marked. At position (5) the second and third formants are drooping downward indicating a bilabial consonant. The faint formant bars following are typical of nasals. Position (5) is the bilabial nasal consonant m. At position (6) the second and third formants are again drooping, indicating another bilabial consonant. The formant lines which follow are immediately strong and no random noise burst is present. These are all signs of the voiced bilabial consonant b as in 'back'. The final position is (8), the velar stop consonant k. The common locus of the second and third formants characteristic of velar consonants is clearly shown.

In general it is possible to learn to identify some key features of speech sounds from spectrographs in a couple of hours, but to arrive at the point where they may be read requires months of daily practice.

Fig. 3.6 Famous phrase, 'We were away a year ago.'

3.4 Applications

The speech spectrograph has made the extensive work in the field of acoustic phonetics possible [1]. The device provides an objective and quantitative analysis of vocal sounds. It is possible to determine the voiced and unvoiced areas of speech along with their frequency composition. Manners of articulation may be discerned and some relative measurements can be made of vowel heights, the degree of front or backness and relative rates of movement of some articulators.

Standard visible references on speech sounds can be made for training phoneticians. These references have been used to provide the feedback needed for hearing impaired individuals to improve their speech [4].

In addition to displaying the speech characteristic necessary to convey meaning a spectrograph shows several speaker dependent traits. This has led to the use of spectrographs as a voiceprint analog of a fingerprint for speaker identification. Though incompletely developed, the method has already been used in hundreds of court cases [10].

Within the engineering field, speech compression and speech

synthesis are natural areas for the application of spectrographic analysis. The technique could objectively identify the distortions characteristic of certain processing, and at the same time determine what speech sound to associate each distortion with.

A spectrographic representation also serves as a basis for some speech recognition algorithms.

Chapter 4

The proposed speech spectrograph system

4.1 Introduction

In reviewing the literature concerning speech spectrographs it is clear that there are two points deserving more attention. The first is that to be useful a spectrograph must be of fairly high quality. By this it is meant that the more subtle features of speech such as fricative consonants must be discernable, not merely vowel formants. The second point is that to be used a spectrograph must be produced quickly. The latter point is valid in an interactive situation which is the most common. Typically an individual will speak into a microphone and wait for the spectrograph. Informal experiments conducted during the course of this work have shown that a delay of about 10 seconds is agreeable but a 40 second delay gets to be irritating after a few trials. Besides the slowed work pace excessive delays make it difficult to associate sounds with their spectrographs. For these reasons careful attention must be paid to what the spectral analysis will resolve and it must be efficiently coded to minimize processing time.

Section 4.2 defines the spectrum of a digital signal in terms of its Z transform and presents a unified equivalence to several filter structures including the DFT. Section 4.3 determines several parameters such as the number of analysis points required to produce a readable speech spectrograph.

Definition of spectrum and analysis structures

Consider a digital sequence $x(n)$. The spectrum of this signal may be defined as its Z transform on some region of interest in the z plane. Figure 4.1 shows two of the possible regions.

$$X(z_k) \triangleq \sum_{n=-\infty}^{\infty} x(n)z_k^{-n} \quad (4.1)$$

Equation 4.1 defines the bilateral Z transform $X(z_k)$, of a discrete signal $x(n)$, at a point z_k in the z plane. The signals, coefficients and transforms are in general complex. In many practical problems the sequence $x(n)$ is quite long and a spectral estimate is obtained by operating on an N term long subsequence of $x(n)$. Where the spectral composition of $x(n)$ is time-varying what may actually be required is a series of spectral estimates displaced in time.

Fig. 4.1 Two possible regions for evaluating a signal spectrum.

$$X_i(z_k) = \sum_{n=0}^{N-1} x(n+i)z_k^{-n} \quad (4.2)$$

Equation 4.2 defines a modified transform $X_i(z_k)$ evaluated over a finite record length N . The parameter i indicates the time of the spectral estimate by denoting the first sample of $x(n)$ used. Expanding eq. 4.2 yields:

$$X_i(z_k) = x(N-1+i)z_k^{-(N-1)} + x(N-2+i)z_k^{-(N-2)} + \dots + x(i)z_k^{-0} \quad (4.3)$$

A FIR filter realization of eq. 4.3 is shown in Fig. 4.2. The boxes containing Z^{-1} denote unit delay elements and the z_k are the complex filter coefficients. A bank of such filters operating in parallel can determine the spectrum at several points z_k simultaneously, providing a series of spectral cross-sections ~~displaced in time~~. When these cross-sections are displaced by a unit delay they are referred to as the sliding spectrum.

An approximate but convenient measure of complexity in DSP algorithms is the number of complex multiplications required. On this basis the complexity of performing a sliding spectral

analysis with the configuration of Fig. 4.2 is near N^2 . This assumes a bank having N filters each N taps long.

Complexity may be reduced by establishing a recursive relationship between $X_i(z_k)$ and $X_{i+m}(z_k)$, where $X_{i+m}(z_k)$ is a spectrum estimate evaluated m unit delays after $X_i(z_k)$. Modifying eq. 4.3:

$$X_{i+m}(z_k) = x(N-1+i+m)z_k^{-(N-1)} + x(N-2+i+m)z_k^{-(N-2)} + \dots + x(i+m)z_k^{-0} \quad (4.4)$$

Comparing 4.3 and 4.4:

$$X_{i+m}(z_k) = z_k^m X_i(z_k) + \underbrace{x(N-1+i-m)z_k^{-(N-1)} + \dots + x(N+i)z_k^{-(N-m)}}_{m \text{ terms}}$$

$$\underbrace{-x(i+m-1)z_k^{-1} - \dots - x(i)z_k^{-m}}_{m \text{ terms}} \quad (4.5)$$

Fig. 4.2 A FIR structure for sliding spectrum analysis at z_k .

Equation 4.5 shows that $2m$ complex multiplications are required to obtain $X_{i+m}(z_k)$ from $X_i(z_k)$. The complexity of a configuration based on eq. 4.5 is therefore only $2m$. At $m = N/2$ there is no saving but at the most efficient case of $m = 1$, eq. 4.5 reduces to;

$$X_{i+1}(z_k) = z_k[X_i(z_k) - x(i)] + x(N+i)z_k^{-(N-1)} \quad (4.6)$$

or on replacing i with $i-1$ and regrouping:

$$X_i(z_k) = z_k[X_{i-1}(z_k) - x(i-1)] + x(N+i-1)z_k^{-N} \quad (4.7)$$

From eq. 4.7 it is seen that only 2 complex multiplications are required to obtain each succeeding spectral estimate. A recursive configuration for eq. 4.7 is shown in Fig. 4.3. Assuming again a sliding spectral analysis based on a bank of N filters yields a complexity of only $2N$ as opposed to N^2 for Fig. 4.2.

A convenient region for performing a spectral analysis is the unit circle. If the z_k are restricted to the values $z_k^n = \exp(j2\pi kn/N)$ the defining eq 4.2 becomes;

$$X_i(\exp(j2\pi k/N)) = \sum_{n=0}^{N-1} x(n+i)\exp(-j2\pi kn/N) \quad (4.8)$$

Fig. 4.3 A recursive structure for spectrum analysis at z_k .

which is the DFT of the sequence $x(i)$ through $x(N-1+i)$ inclusive. Thus the DFT performs a spectral analysis at equally spaced points on the z plane unit circle, and the structures of Fig. 4.2 and Fig 4.3 can implement the DFT.

For the special case of DFT analysis Fig. 4.3 simplifies to the structure of Fig. 4.4. One complex multiplication is required for each succeeding spectral estimate making the bank complexity only N . Fig. 4.5 shows the bank with the large delay z^{-N} common to each element.

This is very interesting because none of the FFT algorithms for evaluating the DFT have a complexity as low as N . For example, radix 2 decimations have an approximate complexity of $N \log(N)/2$. Thus an FFT is less efficient than a filter bank in performing a sliding spectral analysis, provided sufficient spectral cross-sections are required [11].

In some cases the spectrum of $x(n)$ changes slowly and so it is sufficient to calculate a spectral cross-section every m intervals. An FFT performing this type of analysis is called a hopping FFT since it 'hops' over $(m-1)$ samples between determining each spectral cross-section. Its complexity remains $(N/2) \log_2 N$.

Fig. 4.4 A recursive structure for the DFT at $\exp(j2\pi k/N)$.

Fig. 4.5 A bank for the DFT at $\exp(j2\pi k/N)$, $k=0,1,2,\dots,N-1$.

which is very efficient for even small m . Since the recursive structure of Fig. 4.4 cannot hop over data its complexity is effectively mN in a hopping analysis. Equation 4.9 states the criterion by which a hopping spectral analysis is most efficiently performed with an FFT as opposed to a filter bank:

$$m > (1/2)\log_2 N \quad (4.9)$$

where m is hop size in samples, an integer

N is FFT length

Since the DFT is equivalent to a filter bank it is reasonable to look at its impulse and frequency responses. The input-output relation for a linear time-invariant FIR structure is;

$$y(i) = \sum_{n=0}^{N-1} x(n)h(i-n) \quad (4.10)$$

where $h(n)$ is the impulse response

Comparing eq. 4.10 with eq 4.8 for the DFT it is seen $h(i-N)$
 $= \exp(j2\pi k(i-N)/N)$ or:

$$h(n) = \exp(j2\pi kn/N) \quad (4.11)$$

The frequency response is found from the Z transform of the impulse response:

$$H(z) = \sum_{n=0}^{N-1} \exp(j2\pi kn/N) z^{-n} \quad (4.12)$$

$$= \frac{1 - z_u^{-N}}{1 - z_u^{-1}} \quad (4.13)$$

$z_u = \exp(-j2\pi k/N) z$

$$= \frac{1 - z^{-N}}{1 - \exp(j2\pi k/N) z^{-1}} \quad (4.14)$$

On the unit circle $z = \exp(j\omega)$:

$$H(\omega, k) = \frac{1 - \exp(-j\omega N)}{1 - \exp(-j[\omega - 2\pi k/N])} \quad (4.15)$$

or equivalently:

$$H(\omega, k) = \exp(-j\omega[N-1]/2) \exp(-j\pi k/N) \frac{\sin(\omega N/2)}{\sin(\omega/2 - \pi k/N)} \quad (4.16)$$

This response is shown in Fig. 4.6 for $N = 8$. Only the sidelobes of the response for $k = 4$ are shown for clarity, but

each mainlobe has them. On a power scale with the amplitude of the mainlobe at unity the first sidelobe is down 13 dB. The tails pass through zero when $w = 2 k/N$.

There are two problems with this response. These are spectrum leakage and the picket fence effect [12]. The leakage problem is due to the large and slowly diminishing sidelobes which admit components from other bands. The picket fence effect is caused by the sharp taper of the mainlobe. A signal component which is not directly under the peak of a mainlobe is attenuated. The worst case at $w = 3.5$ in Fig. 4.6 is perceived by both responses 3 and 4, but at only 0.6 of its actual amplitude.

Both problems may be reduced by time domain windowing [13], [14]. The idea is to premultiply the time domain sequence $x(n)$ by an appropriate windowing function $w(n)$ before it is transformed, as in eq. (4.17):

$$X_1(\exp(j2\pi k/N)) = \sum_{n=0}^{N-1} x(n+i)w(n)\exp(-j2\pi nk/N) \quad (4.17)$$

The modified impulse response is:

$$hw(n) = w(n)\exp(j2\pi kn/N) \quad (4.18)$$

When $w(n)$ is symmetrical $w(n) = w(-n)$ and the modified frequency response is;

$$H_w(w, k) = H(w, k) * W(w) \quad (4.19)$$

where $H(w, k)$ is defined in eq 4.16, the $*$ operator denotes circular convolution and $W(w)$ is the fourier transform of the window function.

A convenient window is the Hanning window defined by:

$$w_H(n) = 0.5(1 - \cos(2\pi n/N)) \quad (4.20)$$

Its fourier transform is:

$$W_H(w) = 0.5d(w) + 0.5d(w - 2\pi/N) + 0.5d(w + 2\pi/N) \quad (4.21)$$

where $d(\cdot)$ is the delta function

The modified frequency response of the DFT for $k=0$, is [11]:

$$H_w(w, k) = \frac{\sin(wN/2) + 0.5\sin(wN/2 - \pi) + 0.5\sin(wN/2 + \pi)}{\sin(w/2) \sin(w/2 - \pi/N) \sin(w/2 + \pi/N)} \quad (4.22)$$

|
k=0

$H(w,0)$ and the modified $H_w(w,0)$ are shown in Fig. 4.7 for comparison. It can be seen that the mainlobe of the latter is much broader and the sidelobes are not visible because of the linear scale. The amplitude of the first sidelobe is actually at -31 dB, verses only -13 dB for the unwindowed response.

There are several other types of windows well documented in the literature [13],[14]. In general the technique involves a tradeoff between mainlobe width and sidelobe amplitude.

An alternate view of the DFT is to consider it a complex amplitude modulator which produces $x(n)\exp(-j2\pi kn/N)$ from $x(n)$, and then convolves it with a rectangular impulse response. This view is shown in Fig. 4.8. As k ranges from $[0, N-1]$ the spectrum of $x(n)$ is shifted left past a fixed lowpass filter as shown in Fig. 4.9. The filter response is $H(w,0)$ from eq 4.16 or $H_w(w,0)$ from eq. 4.22 if a Hanning window is applied.

4.3 Analysis parameters for a speech spectrograph

The rationale for the selection of certain analysis parameters is difficult to construct without the visual feedback of a

Fig. 4.6 DFT filter bank response with $N=8$

Fig. 4.7 Effect of a Hanning window on DFT frequency response

Fig. 4.8 Complex scanning filter view of DFT.

Note: Only one period of the spectrum is shown for clarity.

Fig. 4.9 Spectrum illustrations for Fig. 4.8.

functioning spectrograph system. The approach taken here is to determine reasonable boundary values based on the characteristics of speech sounds and then chose slightly more conservative values based on some other criteria. Examples of parameters discussed are the sampling rate and the number of analysis points.

In any time-frequency analysis there must always be a tradeoff in resolution between the two. For this reason both wideband and narrowband spectrographs are studied. The wideband spectrograph has the best time resolution to determine the location of formants but not their fine structure. The pitch of the vocal wave is usually visible as a series of vertical striations. Narrowband spectrographs can determine fine structure causing formants to appear as a band of closely spaced horizontal striations at the cost of smearing the time axis. The interest here is in wideband spectrographs because some individuals have learned to read them.

Sampling rate - Most speech energy is concentrated under 3 khz but in identifying certain consonants it is helpful to perceive some higher frequencies. A 5 khz bandwidth provides a small margin and indicates a nominal 10 khz sampling rate.

Number of quantization bits - The standard A/D converters are 8 and 12 bits. Sixteen bits is available at a premium.

Signal/noise ratio due to quantization is approximately:

$$S/N \text{ dB} = 6b - 1.24 \quad (4.22)$$

assuming a speech signal and low probability of clipping [15].

The 'b' is the number of bits in the quantizer. Two or three bits are easily lost in processing resulting in a low S/N of only 28 dB. The dynamic range of speech is in excess of 60 dB. A 12 bit A/D provides a low S/N of 52 dB economically. In practice the 52 dB figure is seldom realized because of the background noise in most rooms.

Number of analysis points - On a spectrograph the lower three formants move about quite freely and sometimes even merge as in the case of velar consonants or the approximant 'r'. However it is useful to be able to recognize the standard vowels as three separate formants. Figure 4.10 is a table listing the formant frequencies of 7 English vowels.

Fig. 4.10 Formant frequencies of 7 English vowels.

• Key word	F1	F2	F3
heed	280 hz	2250 hz	2890 hz
hid	400	1920	2560
head	550	1770	2490
had	690	1660	2490
hod	710	1100	2540
hood	450	1030	2380
food	310	870	2250

The minimum frequency spacing between any two formants of a vowel is about 300 hz. To be able to follow the motion of the formants indicates a spacing between analysis points of $300/2 = 150$ hz maximum. Allowing for overlap of adjacent filter bands and some more resolution suggests a spacing of about $150/2 = 75$ hz. With a 5 khz bandwidth this implies $5000/75 = 66$ analysis points.

Sliding or hopping analysis - A sliding analysis requires the most computation and so is to be avoided if possible. In short-time analysis methods based on vocal tract models it is

usually found that a 50 to 100 hz sampling rate is sufficient for the model parameters [16]. This implies the vocal tract changes little in a $1/100 = 10$ ms interval. Allowing some margin for distortion at 10 ms sampling suggests a $10/2 = 5$ ms interval between outputs from the filter bank. A sliding analysis provides updates every input sample interval or $1/10,000 = 0.1$ ms. Compared to 5 ms the 0.1 ms seems much too conservative and consequently a hopping type analysis is chosen.

Length of FFT - With the choice of a hopping analysis it is likely that a hopping FFT is the most efficient structure to perform it. The decision depends in part on the length N of the FFT. The most time efficient FFT's coded are based on radix-4 decimations. Thus it is convenient if the length N is a power of 4. Since speech data is a real function its DFT is complex conjugate antisymmetric. This means nearly half the frequency points are redundant and a 130 pt. transform is needed to get 66 unique frequency pts. Choosing 65 analysis pts. would allow use of a 128 pt. real DFT. This can be constructed from a 64 pt. complex FFT and a doubling algorithm. The number $64 = 4^3$, thus the radix 4 decimation can be used with $N=64$. The input speech data sequence, since it is real, will be 128 samples long. The FFT and doubling algorithm are discussed further in chapter 5.

Structure for analysis - Having chosen a hopping analysis and determined an FFT length N , eq. 4.9 is used to determine if the FFT is the most efficient structure. Repeating eq. 4.9:

$$m > (1/2)\log_2 N \quad (4.9)$$

The result is the hopping FFT is most efficient for m greater than 3. A hop of $m = 3$ is only $(3)(0.1) = 0.3$ ms, substantially less than the 5 ms considered and so the hopping FFT is chosen to perform the spectral analysis.

Hop size - A lower boundary for the hop size is set by the FFT. A 128 pt. data sequence is $128/10,000 = 12.8$ ms long. It is unlikely a hop size less than 10 % of this or approximately 1.3 ms would cause much change. The hop size should probably be between 1.3 and 5 ms. A check of the graphics modes available for standard monitors indicates a best flicker-free horizontal by vertical resolution of $(600_H)(200_V)$. Many phrases and short sentences are uttered in under 2 seconds. It turns out that if the hop size m is selected on the basis of having 600 spectral cross-sections derived from 2 seconds of data, then $m = 33$ samples or $33/10,000 = 3.3$ ms, and conveniently $1.3 < 3.3 < 5$ ms.

The chosen hop size is then $m = 33$ samples.

Dynamic range compression - The dynamic range at the output of the FFT could possibly be $(2^{15}-1)$ with a 16 bit processor, but a standard monitor has at best a 16 level gray scale. A dynamic range compression algorithm must map $(2^{15}-1)$ into 2^4 levels while favoring moderately weak signals to render consonants visible. At the same time the very weakest signals must be removed to reduce background noise.

Chapter 5

Coding the FFT

5.1 Introduction

The FFT is the time and space critical number crunching routine of the spectrograph software package. Therefore considerable effort was expended to efficiently code a radix 4 decimation algorithm. A radix 4 FFT was chosen because it has been shown [17] repeatedly that it is the fastest method of computing the DFT.

Another reason for the effort was the academic challenge. The execution time of an FFT is a widely recognized benchmark for both DSP processors and software. The 64 point complex FFT in particular has been emerging as a standard for comparing microprocessors, including the Texas Instruments TMS32010 used in this work. Thus established figures are available for a routine which must now be coded and it becomes very difficult not to compete with them.

The FFT described here is not the same FFT as was coded for

the spectrograph project. Time constraints necessitated development cease when a reasonable 544 usec was achieved. This marked an improvement over the 783 usec announced by Texas Instruments in [18] but finishes second to the 535 usec announced by DSPS [19].

After completing the spectrograph it was decided to return to the problem and design FFT code which would at least match the fastest benchmark known. It is the results of this latter effort which are discussed here.

The approach taken is quite logical. First, the FFT is examined in section 5.2 with the help of a flow diagram representation. The purpose of the flow diagram is to represent the algorithm visually in such a way that short-cuts and redundant paths become more obvious. Then several inferences are drawn as to how the software could best be coded. Secondly, the architecture of the TMS32010 is considered in section 5.3. Again several inferences are drawn on how standard DSP operations should best be coded.

Then in section 5.4 through 5.6 the results of sections 5.2 and 5.3 are used in coding the FFT. The intent of the algorithm and architectural examinations is to reduce as much as possible

the search space in the practical coding problem. The idea is that the optimized code should be a logical consequence of the algorithm and the processor.

Section 5.7 derives the ordering of the FFT input sequence and presents means to accomplish it. Section 5.8 presents a development of the doubling algorithm used to obtain the DFT of a 128 pt. real data sequence from the 64 pt. complex DFT. In the final section, 5.9, a benchmark comparison is made between the FFT code of this work and other sources.

5.2 The flow diagram

5.2.1 Deriving the flow diagram

The flow diagram is derived by a direct examination of the DFT equation. The procedure is a generalization of that used in [15] for deriving the radix 2 structure.

The DFT equation is:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad (5.1)$$

where $N = 64$ and $W_N^{nk} = \exp(-j2\pi/N)nk$

The sequence $x(n)$ can be decimated into four subsequences:

$$x(n) = x(4i) + x(4i+1) + x(4i+2) + x(4i+3) \quad (5.2)$$

where $i = [0, 15]$

Rewriting $X(k)$ in terms of these decimations:

$$\begin{aligned} X(k) = & \sum_{i=0}^{N/4-1} x(4i) W_N^{4ik} + \sum_{i=0}^{N/4-1} x(4i+1) W_N^{(4i+1)k} \\ & + \sum_{i=0}^{N/4-1} x(4i+2) W_N^{(4i+2)k} + \sum_{i=0}^{N/4-1} x(4i+3) W_N^{(4i+3)k} \end{aligned} \quad (5.3)$$

$$\begin{aligned} = & \sum_{i=0}^{N/4-1} x(4i) W_N^{4ik} + W_N^k \sum_{i=0}^{N/4-1} x(4i+1) W_N^{4ik} \\ & + W_N^{2k} \sum_{i=0}^{N/4-1} x(4i+2) W_N^{4ik} + W_N^{3k} \sum_{i=0}^{N/4-1} x(4i+3) W_N^{4ik} \end{aligned} \quad (5.4)$$

and noting $W_N^{pik} = W_{N/p}^{ik}$:

$$\begin{aligned} X(k) = & \sum_{i=0}^{N/4-1} x(4i) W_{N/4}^{ik} + W_N^k \sum_{i=0}^{N/4-1} x(4i+1) W_{N/4}^{ik} \\ & + W_N^{2k} \sum_{i=0}^{N/4-1} x(4i+2) W_{N/4}^{ik} + W_N^{3k} \sum_{i=0}^{N/4-1} x(4i+3) W_{N/4}^{ik} \end{aligned} \quad (5.5)$$

The sequence $X(k)$ is now seen to be the sum of 4 sixteen-point transforms. If each 16 point transform is given a name $G_i(k)$, then $X(k)$ may be written as:

$$X(k) = G_0(k) + W_N^k G_1(k) + W_N^{2k} G_2(k) + W_N^{3k} G_3(k) \quad (5.6)$$

Note since the sequence $X(k)$ is 64 long and the $G(k)$ are only 16 long, the periodic nature of the $G(k)$ is important:

$$G(k) = G(k+16p) \quad (5.7)$$

where p is an integer

To illustrate, consider the case of $k = 1$:

$$G(1) = G(17) = G(33) = G(49) \quad (5.8)$$

This indicates that values of $X(k)$ spaced 16 apart will all be constructed from the same $G_0(1)$, $G_1(1)$, $G_2(1)$ and $G_3(1)$ terms. Solving for $k = 1, 17, 33$ and 49:

$$X(1) = G_0(1) + W_N^1 G_1(1) + W_N^2 G_2(1) + W_N^3 G_3(1) \quad (5.9)$$

$$X(17) = G_0(1) + W_N^{17} G_1(1) + W_N^{34} G_2(1) + W_N^{51} G_3(1)$$

$$X(33) = G_0(1) + W_N^{33} G_1(1) + W_N^{66} G_2(1) + W_N^{99} G_3(1)$$

$$X(49) = G_0(1) + W_N^{49} G_1(1) + W_N^{98} G_2(1) + W_N^{147} G_3(1)$$

Noting $W_N^P = W_N^{P-64} W_N^{64} = W_N^{P-64}$ and $W_N^{16} = -j$, (5.10)

$$W_N^{32} = -1, W_N^{48} = j, \text{ for } N=64, \text{ then:}$$

$$X(1) = G_0(1) + 1W^1 G_1(1) + 1W^2 G_2(1) + 1W^3 G_3(1) \quad (5.11)$$

$$X(17) = G_0(1) - jW^1 G_1(1) - 1W^2 G_2(1) + jW^3 G_3(1)$$

$$X(33) = G_0(1) - 1W^1 G_1(1) + 1W^2 G_2(1) - 1W^3 G_3(1)$$

$$X(49) = G_0(1) + jW^1 G_1(1) - 1W^2 G_2(1) - jW^3 G_3(1)$$

Similarly:

$$X(2) = G_0(2) + 1W^2 G_1(2) + 1W^4 G_2(2) + 1W^6 G_3(2) \quad (5.12)$$

$$X(18) = G_0(2) - jW^2 G_1(2) - 1W^4 G_2(2) + jW^6 G_3(2)$$

$$X(34) = G_0(2) - 1W^2 G_1(2) + 1W^4 G_2(2) - 1W^6 G_3(2)$$

$$X(50) = G_0(2) + jW^2 G_1(2) - 1W^4 G_2(2) - jW^6 G_3(2)$$

It is important to note the pattern for combining the $G(k)$ into $X(k)$ is the same in each case. Using the notation of [11]

the pattern can be represented by:

and in general:

for $k = [0, 15]$.

The decimation to radix 4 may be completed by writing each of the $G(k)$ as four 4 point transforms. For example:

$$G_0(k) = \sum_{i=0}^{N/4-1} x(4i) w_{N/4}^{4ik} \quad (5.13)$$

$$G_0(k) = \sum_{r=0}^{N/4-1} x(16r) w_{N/4}^{16rk} + \sum_{r=0}^{N/4-1} x(16r+4) w_{N/4}^{(16r+4)k} \quad (5.14)$$

$$+ \sum_{r=0}^{N/4-1} x(16r+8) w_{N/4}^{(16r+8)k} + \sum_{r=0}^{N/4-1} x(16r+12) w_{N/4}^{(16r+12)k}$$

$$\begin{aligned}
 G_0(k) = & \sum_{r=0}^{N/16-1} x(16r)w_{N/16}^{rk} + w_{N/4}^k \sum_{r=0}^{N/16-1} x(16r+4)w_{N/16}^{rk} \\
 & + w_{N/4}^{2k} \sum_{r=0}^{N/16-1} x(16r+8)w_{N/16}^{rk} + w_{N/4}^{3k} \sum_{r=0}^{N/16-1} x(16r+12)w_{N/16}^{rk}
 \end{aligned} \quad (5.15)$$

If the 4 point transforms are named $H_i(k)$:

$$G_0(k) = H_0(k) + w_{N/4}^k H_1(k) + w_{N/4}^{2k} H_2(k) + w_{N/4}^{3k} H_3(k) \quad (5.16)$$

The $H(k)$ will have period 4 resulting in the following pattern for forming $G(k)$:

This pattern of combinations is the same pattern derived for the construction of $X(k)$ from the $G(k)$. As might be expected, the patterns of combinations for constructing $G_1(k)$, $G_2(k)$, and $G_3(k)$ are of this form also.

At this point all that remains in deriving the 64 point FFT flow diagram is to discern the structure of the basic 4 point DFT building block.

Considering the equation for the 4 point DFT:

$$X(k) = \sum_{n=0}^3 x(n)W_4^{nk} \quad (5.17)$$

expanding:

$$X(0) = x(0) + 1x(1) + 1x(2) + 1x(3) \quad (5.18)$$

$$X(1) = x(0) - jx(1) - 1x(2) + jx(3)$$

$$X(2) = x(0) - 1x(1) + 1x(2) - 1x(3)$$

$$X(3) = x(0) + jx(1) - 1x(2) - jx(3)$$

$$\text{where } W_4^0 = 1, W_4^1 = -j, W_4^2 = -1 \text{ and } W_4^3 = j \quad (5.19)$$

It is interesting to note that the pattern of combinations involved in performing a 4 point DFT is the very same pattern used to combine small DFT's into large DFT's. Thus the 4 point DFT is also represented by:

A 4 point DFT

The entire 64 point FFT has been reduced to a sequence of complex multiplications and repeated applications of the above pattern. This is shown in the flow diagram of Fig. 5.1.

5.2.2 Observations

The highly regular structure of the algorithm is clear from Fig. 5.1. It may be seen to consist of a series of 48 4 point DFT's which are sometimes separated by complex multiplications, only 76 of which are non-trivial. Also, it should be possible to separate the complex multiplications at each stage from the 4 point DFT's. Then, in this simplest case, only two code modules need be produced. One would perform a 4 point DFT and the other a complex multiplication. The first column of the transform could be coded by invoking the 4 point DFT module 16 times with appropriate address changes. Several calls to the multiplication module with the appropriate coefficients would complete the manipulations required to begin the second column. Then the second and third columns would be coded in similar fashion.

Implicit in the above is the notion that the entire transform may be performed one column at a time. While this is true and

Fig. 5.1 FFT flow diagram.

is the approach used here, it should be mentioned that there are other possibilities particularly for multiaccumulator machines. However for the single accumulator processor the advantages are essentially nil, and so they are not discussed.

The alternative to handling the 4 point DFT's and complex multiplications separately is to combine them in either of two ways. Since it is possible to separate the complex multiplications from the 4 point DFT's it follows that when combining them the multiplications could be combined with either the preceding or following DFT's. These two alternatives are shown in Fig. 5.2.

For convenience the 4 point DFT's which occur so frequently in this discussion will be referred to in short form by the equivalent name R4B, for Radix 4 Butterfly.

The left hand R4B of Fig. 5.2 is normally associated with a DIT (decimation in time) structure and the right hand R4B with a DIF (decimation in frequency) structure. This distinction as to their origins seems rather superfluous since both arise from either decimation by a minor shift of the observer's viewpoint. However since the convention is common it is used here.

Fig. 5.2 4 point DFT's.

Computationally there is no fundamental difference between the DIT and DIF R4B's. That is to say an FFT constructed from either decimation will have an identical number of complex multiplications and additions. If the calculations are performed accurately results will also be identical. On the other hand processor architecture can be more suited to one than the other and differences in the numbers and types of R4B's required exist at the coding level. Architecture and coding are discussed below.

5.3 Architectural considerations

5.3.1 An introduction to the TMS32010 architecture

A programmers view of the TMS32010 architecture is shown in Fig. 5.3. It has several features which make it especially useful in DSP applications.

There is a 16 bit parallel multiplier capable of forming a 31 bit product. Data, 16 bits wide, can be scaled on moving it into the ALU or out from the ACC by two shifters without any time

Fig. 5.3 Block diagram of the TMS32010.

overhead. Two auxiliary registers with autoincrement and autodecrement by one modes may be used as pointers or loop counters and the ARP selects the active register at any time.

The basic architecture of the machine is a modified Harvard, program memory is external (maximum 4 kwords) and distinct from data RAM. Unfortunately there are only 144 words of data RAM but this is partially compensated for by an immediate addressing mode instruction and the ability to move constants from program memory into data RAM. This latter is a very slow mode of data access and so generally avoided in high speed computation. The advantage of separating program and data memory is that it allows pipelining for considerable speed. Most instructions execute at a 5 Mhz rate.

The data RAM is divided into two pages because of the small instruction address field. Page 0 contains 128 words and page 1 the remaining 16. A one bit DP register selects the active page.

A four level stack allows a subroutine and interrupt facility. Communication with the outside world is provided for by means of the BIO, a conditional branch pin, INT a single vectored interrupt, RS a reset pin, and eight bidirectional

ports. The data bus is brought out of the chip for use with the port commands. External data is moved directly in and out of the data RAM only.

5.3.2 Implications

Several conclusions may be drawn directly from the TMS32010 architecture as to how the coding should be approached. The 64 complex pt. FFT requires a minimum of 128 data locations assuming the same locations are used for the output data as for the input data. Fast code requires this option which means the R4B's must perform their computations effectively in place. The word 'effectively' is used because it is sufficient that the output data points of each R4B overwrite the input data points. Temporary locations may be used to store intermediate values during an R4B computation. If it is not possible to overwrite any of the current R4B locations the temporaries will have to be on the second page of data memory. All other locations on the first page will contain data needed by other R4B's and cannot be overwritten. The addresses on the first page which any R4B must access will not necessarily be sequential and generally differ from each other by a constant larger than one. Thus the immediate address mode is the only efficient scheme because the

auxiliary registers can only autoincrement or autodecrement by one. For temporary data locations on the second page the case is different. These may be grouped into adjacent locations making indirect addressing by the auxiliary registers practical. It is also best since moving data to the second page with direct mode addressing would waste time continually setting and resetting the data page register.

For the 76 complex coefficients immediate mode addressing is the only choice because there is insufficient data memory for their storage. Run time calculation or program memory transfers are considerably slower.

Overhead on loop counting may be eliminated by eliminating loops altogether through inline coding. The presence of the ample 4k program memory address space makes this practical.

Finally it is useful to consider the configuration of the arithmetic elements of the TMS32010, again in Fig. 5.3. Products are formed in the 32 bit P register from two factors one of which is in the 16 bit T register. The other factor comes from either the data memory and is also 16 bits, or from an immediate mode instruction and is just 13 bits. In either case, the T register is loaded from data memory and feeds into the parallel

multiplier. The P register feeds into the ALU through a multiplexer. The 32 bit ALU in turn feeds into the 32 bit accumulator. Contents of the accumulator may be stored back in data memory with or without left shifting of its contents.

The sequence cannot be reversed. Nor is it possible to store the T register directly in data memory. It is a poor temporary register. There is no direct path from the accumulator back to the T register. Thus the structure has a built in bias towards calculations of the multiply accumulate form as opposed to an accumulate and multiply form. An example of each is given below:

Multiply accumulate

$$(a_1 b_1) + (a_2 b_2) = MA \quad (5.20)$$

Accumulate and multiply

$$(a_1 + b_1)(a_2 + b_2) = AM \quad (5.21)$$

Assuming the a's are variables and the b's are constants an instruction sequence for eq. 5.20 is:

```
LT    a1
MPYK  b1
PAC
LT    a2
MPYK  b2
APAC
SACH  MA,4
7 instructions
```

Sequence for eq. 5.20

The LT a_1 means load the T register with the contents of data memory location a_1 . MPYK b_1 multiplies the T register by the immediate constant b_1 and puts the product into the P register. PAC moves the P register into the accumulator. Similarly, the next two instructions form the product of a_2 and b_2 in the P register. Next is APAC. APAC adds the contents of the P register to the accumulator instead of merely overwriting them as does PAC. The last instruction, SACH MA,4 stores the final result from the high word of the accumulator into data memory location MA with a left shift of 4.

Following is a sequence for eq. 5.21. Here it is assumed that both the a's and b's are variables because the TMS32010 instruction set does not include an add immediate instruction. The ADD instruction used adds the contents of the indicated memory location to the accumulator and the LAC instruction loads the accumulator with the contents of the indicated memory location. The 15's are left shifts for scaling. MPY is like the MPYK instruction except that its operand is a variable.

```

LAC   a1,15
ADD   b1,15
SACH  Temp1
LAC   a2,15
ADD   b2,15
SACH  Temp2
LT    Temp1
MPY   Temp2
PAC
SACH  AM,1

```

10 instructions

Sequence for eq. 5.21

Comparing the instruction sequences for eq. 5.20 and eq. 5.21 it is seen that the multiply accumulate operation requires only 70 % of the code (and time) needed for the accumulate and multiply operation. Clearly a careful ordering of arithmetic operations is important for efficient code.

Another instruction which is very useful is the LTA mem instruction. LTA is a combination of LT and APAC. Specifically, LTA loads the T register with the contents of an indicated memory location and at the same time adds the contents of the P register to the accumulator. It can be used to eliminate all the APAC instructions except for the last in a sequence of multiply accumulates. For example consider eq. 5.22.

$$(a_1b_1) + (a_2b_2) + (a_3b_3) = MA2 \quad (5.22)$$

Equation 5.22 has the same form as eq. 5.20 but has an extra term. An instruction sequence for eq. 5.22 is shown following:

LT	a_1
MPYK	b_1
PAC	
LT	a_2
MPYK	b_2
LTA	a_3
MPYK	b_3
APAC	
SACH	$MA2,1$

9 instructions

One APAC instruction has been saved in this example. With more terms more APAC's could be saved and so the utility of the LTA instruction increases with the number of terms in the multiply accumulate operation. The multiply accumulate operation is basic to the calculation of complex products needed by the FFT algorithm. Earlier it was pointed out that while there is no inherent computational difference between the DIT and DIF decimations of the DFT, a difference can exist at the coding level because of a particular processor architecture. Returning to this point it is helpful to again consult Fig. 5.1. Looking at Fig. 5.1 as a DIT flow diagram three distinct types of DIT-R4B can be identified. There are 21 R4B's having no nontrivial complex multiplications. There are 5 R4B's having 2 complex multiplications and there are 22 R4B's having 3 complex multiplications. These will be referred to as DIT0, DIT2, and DIT3 type R4B's respectively. However, if Fig. 5.1 is regarded as a DIF flow diagram things are a little different. There are now 9 R4B's of another type having 4 complex multiplications each.

This type will be referred to as DIF4. Of the others, there are 24 of DIF0, 5 of DIF2, and 10 of type DIF3. These observations are summarized in the table below.

Fig. 5.4 Comparing the DIT and DIF viewpoints

Number of complex multiplications	Type	
	DIT	DIF
0	21	24
2	5	5
3	22	10
4	+ 0	+ 9
	= 48	= 48

As it must be, the total number of complex multiplications is the same in either case but their distribution amongst the DIF R4B's can be used to advantage. From the previous discussion on use of the LTA instruction it follows that the DIF decimation will save 9 APAC's because of the DIF4 R4B's as compared to the DIT decimation. While this is a very small difference considering the algorithm's total size, it is interesting to note that a difference does exist and so for interest's sake the DIF decimation is used in coding the transform. Examples of each DIF type R4B are shown in Fig. 5.5.

DIF0-R4B

DIF4-R4B

DIF3-R4B

DIF0-R4B

Fig. 5.5 DIF-R4B examples

5.5 The basic DIF0-R4B

Many conclusions on how to code the FFT have already been drawn from the preceding examination of the flow diagram of Fig. 5.1 and the TMS32010 processor architecture. This has considerably reduced the problem search space. What remains is to find a highly efficient means of coding the basic R4B of type DIF0. Since the other three types of DIF R4B's have the same basic pattern of combinations with the addition of some complex multiplications at the output, their coding is also aided by studying the DIF0-R4B.

It has been shown that calculation of each R4B must be essentially in-place. The most straight-forward way to accomplish this is to make the calculation completely in-place, that is without using any temporary locations at all.

The symbol used previously for the basic R4B is repeated below and with it the 4 complex equations it represents.

$$\begin{aligned}
 X(0) &= x(0) + 1x(2) + 1x(1) + 1x(3) \\
 X(1) &= x(0) - 1x(2) - jx(1) + jx(3) \\
 X(2) &= x(0) + 1x(2) - 1x(1) - 1x(3) \\
 X(3) &= x(0) - 1x(2) + jx(1) - jx(3)
 \end{aligned}
 \tag{5.23}$$

Each of the above equations is in complex form. Arithmetic operations on the TMS32010 are real, so it is useful to write the above in terms of the real operations which must be performed. Subscripts r and i indicate the real and imaginary components respectively. Brackets are eliminated for clarity.

$$X_{0r} = x_{0r} + x_{2r} + x_{1r} + x_{3r} \quad (5.24)$$

$$X_{0i} = x_{0i} + x_{2i} + x_{1i} + x_{3i}$$

$$X_{1r} = x_{0r} - x_{2r} + x_{1i} - x_{3i}$$

$$X_{1i} = x_{0i} - x_{2i} - x_{1r} + x_{3r}$$

$$X_{2r} = x_{0r} + x_{2r} - x_{1r} - x_{3r}$$

$$X_{2i} = x_{0i} + x_{2i} - x_{1i} - x_{3i}$$

$$X_{3r} = x_{0r} - x_{2r} - x_{1i} + x_{3i}$$

$$X_{3i} = x_{0i} - x_{2i} + x_{1r} - x_{3r}$$

An attempt to directly calculate the X's from the equations in their present form will be found impossible. This is because of the need to have the results occupy the same locations as the input data. For example, suppose X_{0r} is currently in the accumulator from having just added x_{0r} , x_{2r} , x_{1r} and x_{3r} . It must now be stored in one of the input x locations before computation may proceed. It does not matter which x location is chosen the consequence is the same. Suppose X_{0r} is stored in

the x_{0r} location for convenience. Now x_{0r} has been overwritten and is no longer available for the calculation of x_{1r} , x_{2r} , and x_{3r} . While the problem may still be circumvented at this initial stage, a point is shortly reached where it is not possible to proceed because still-needed data has been overwritten.

The solution is to split the R4B into two stages. In the first stage a set of intermediate values which require access to fewer data points will be calculated. Then the overwriting of data will be less of a problem. Then in the second stage the intermediate values are used without recourse to the originals no longer available to complete the computation.

Looking at the 8 real equations for the X's it is seen that there are several pairs of x's which occur repeatedly. The equations are rewritten below to show this grouping explicitly.

$$\left. \begin{aligned} x_{0r} &= (x_{0r} + x_{2r}) + (x_{1r} + x_{3r}) \\ x_{2r} &= (x_{0r} + x_{2r}) - (x_{1r} + x_{3r}) \end{aligned} \right\} \quad (5.25)$$

$$\left. \begin{aligned} x_{0i} &= (x_{0i} + x_{2i}) + (x_{1i} + x_{3i}) \\ x_{2i} &= (x_{0i} + x_{2i}) - (x_{1i} + x_{3i}) \end{aligned} \right\} \quad (5.26)$$

$$\left. \begin{aligned} x_{1r} &= (x_{0r} - x_{2r}) + (x_{1i} - x_{3i}) \\ x_{3r} &= (x_{0r} - x_{2r}) - (x_{1i} - x_{3i}) \end{aligned} \right\} \quad (5.27)$$

$$\left. \begin{aligned} x_{1i} &= (x_{0i} - x_{2i}) + (x_{3r} - x_{1r}) \\ x_{3i} &= (x_{0i} - x_{2i}) - (x_{3r} - x_{1r}) \end{aligned} \right\} \quad (5.28)$$

A pattern is emerging. All the X 's may be grouped into pairs of the form $X_n = a+b$, $X_{n+2} = a-b$. The following symbol will be used to depict this pattern.

In Fig. 5.6 a flow diagram which summarizes the grouped equations is shown.

Fig. 5.6 Flow diagram for R4B grouped equations.

It is apparent from Fig. 5.6 that the same pattern which is used to calculate the X 's from the intermediate values may also be used to calculate the intermediate values from the input data. For example:

When calculation of the intermediate values is combined with Fig. 5.6, a complete flowgraph for the basic R4B results. This is shown in Fig. 5.7. There is an interesting observation to be made here. The little two input two output pattern used repeatedly in Fig. 5.7 is the symbol of the radix 2 two point butterfly. The R4B has been constructed from radix two butterflies. If each basic R4B of Fig. 5.1 is constructed from the flowgraph of Fig. 5.7 the result is a 64 complex point DFT constructed entirely of radix 2 butterflies. If the effects on data ordering are followed through it turns out that the input sequence ordering is bit reversed exactly as results form a radix 2 decimation. But, the number of complex multiplications required is only 76, not the 98 required for a conventional radix 2 decimation.

Consider the coding of a radix 2 butterfly. In the worst case

Fig. 5.7 A flow diagram for the basic DIF0-R4B

it is possible that an output value will be double the maximum input. Overflow can occur. To guard against it a divide by 2 will be included in the calculation. This is a standard procedure on fixed point machines.

```

LAC   x0r,15   radix 2 butterfly coding
ADD   x2r,15
SACH  x0r      x0r <- (x0r+x2r)/2
SUBH  x2r
SACH  x2r      x2r <- (x0r-x2r)/2

```

The short code segment above shows efficient calculation of two intermediate values needed in Fig. 5.7 from two data values. It demonstrates that calculation of intermediate values can be done in-place and with proper scaling. Only five instructions are required because the accumulator is used to hold a partial result. The calculation is effectively:

- 1) $ACC \leftarrow (x_{0r} + x_{2r}) 2^{15}$
- 2) store $ACC/2^{16}$ in x_{0r} , $x_{0r} \leftarrow (x_{0r} + x_{2r})/2$
- 3) $ACC \leftarrow ACC - x_{2r} 2^{16}$
- 4) store $ACC/2^{16}$ in x_{2r} , $x_{2r} \leftarrow (x_{0r} - x_{2r})/2$

As mentioned earlier the 15 in the LAC and ADD instructions causes a left shift of 15. The SACH is a store instruction with an implicit divide by 2^{16} . This is the source of the divide by 2 to prevent overflow.

In the third line the original value for x_{0r} is overwritten and so it cannot be accessed again. However this problem is circumvented by operating directly on the accumulator. A SUB $x_{2r}, 15$ after the third line would leave the original $x_{0r} 2^{15}$. But since what is actually required in the accumulator is $(x_{0r} - x_{2r}) 2^{15}$ it is better to perform a SUBH x_{2r} which is the equivalent of two SUB $x_{2r}, 15$ instructions. A final SACH x_{2r} completes the butterfly.

There are seven other identical butterflies in Fig. 5.7. Only the addresses change. Five instructions per butterfly times 8 butterflies results in 40 instructions for the basic R4B. This is not too bad in terms of early benchmarks for the processor but it was derived with the unnecessary constraint of no extra temporary memory locations for intermediate data. By relaxing this restriction it should be possible to reduce the instruction count further.

To identify a goal, consider an unrestricted case. Suppose there were a great deal of data memory and that the calculation of an R4B did not have to be performed in-place at all. What is the best that could be done?

The time consuming operations in any algorithm consist of

two types. These are arithmetic operations and data movement operations. The number of arithmetic operations has already been pared down considerably by the use of a radix 4 decimation and the identification of trivial complex multiplications. It is therefore useful at this point to concentrate on data movement.

The code segment for a real radix 2 butterfly confirms this. Of the 5 instructions, only two are actually arithmetic. The other three are just moving data around. It is useful to identify why so much data movement is taking place. The culprits are the intermediate values. Every time an intermediate value is calculated it is also stored. Later it must be accessed, sometimes with an arithmetic operation but sometimes with a data movement instruction. If intermediate data values did not have to be stored data movement could be reduced and some instructions saved. One way to avoid having to store intermediate values is to avoid calculating them. This is feasible with ample memory.

Consider the following code segment for X_{0r} :

LAC	$x_{0r}, 14$	
ADD	$x_{2r}, 14$	
ADD	$x_{1r}, 14$	
ADD	$x_{3r}, 14$	calculation of x_{0r}
SACH	x_{0r}	

5 instructions total

There are five instructions. There is no way around this since it is the initial calculation. Four data points must be accessed and one result stored. The sequence represents the minimum code required. However, X_{2r} differs from X_{0r} only in the sign of two data points, x_{1r} and x_{3r} . Thus it is possible to get X_{2r} with two operations on the accumulator and one more to store the result. This is shown below.

LAC	$x_{0r}, 14$	
ADD	$x_{2r}, 14$	
ADD	$x_{1r}, 14$	
ADD	$x_{3r}, 14$	
SACH	X_{0r}	
SUB	$x_{1r}, 15$	calculation of X_{0r} and X_{2r}
SUB	$x_{3r}, 15$	
SACH	X_{2r}	

8 instructions total

This is the minimum number of instructions required to calculate two of the X's since no two X's differ by less, though there are three other pairs which differ by the same amount.

When all eight X's are calculated as four pairs in this way, the total is 8 instructions per pair times 4 pairs per R4B = 32 instructions for the basic R4B. Unfortunately this goal is unattainable because of the memory constraint but following is an effort to approach it very closely.

It is reasonable to expect the attainable number of instructions for the basic R4B to be somewhere in the range

2

of/de

2

of 32-40. A practical memory constraint is to allow storage of a few intermediate values on the second page of data RAM but still require the final X's to occupy the original data locations so that the calculation is effectively in-place. The compromise is to calculate and store some intermediate values but not all that were used in the 40 instruction version. The intermediate values referred to are the inputs to the radix 2 butterflies of Fig.

5.7. Recalling the code for a radix 2 butterfly, the first thing done is to load the accumulator with the upper input of the butterfly. If this input were never stored it would not have to be reloaded. The next step is to take the lower butterfly input and then add and subtract it from the accumulator with appropriate scaling and intermediate stores. To do this it must still be available as a precalculated group. Thus a more efficient procedure is to precalculate only the inputs for the lower portion of each butterfly, a total of 4, and store them in temporary locations. Because they are calculated in pairs by 5 instructions each, this will take 10 instructions. When all 8 inputs are available in precalculated form, the butterflies of Fig. 5.7 require 5 instructions each for a sum of 20 instructions more. It is not so easy here because the upper inputs are not precalculated and so 2 instructions (one per data value) are needed to access them instead of one. The consequence is each butterfly now takes 6 instructions instead of five for a sum of

24. The total then is $10+24 = 34$ instructions for the basic R4B. Code for this R4B is shown in Fig. 5.8.

Referring to this figure, there are a couple of points worth mentioning. There are 6 butterfly sections separated by horizontal lines. The ordering the first two is arbitrary since they write into temporary locations. The ordering of the remainder must be chosen carefully because their outputs overwrite input data. The ordering is chosen such that data which is overwritten by one section is not required later by another. It is the presence of some temporary locations for intermediate values which makes this close packing possible. Concerning the temporary locations themselves, they are referred to as Temp1, Temp2, etc. This implies that temporary data locations are available on the first page which of course is not so. As discussed previously the proper place for the temporary locations is on the second page of data memory, addressed by the auxiliary registers. This was not done in Fig. 5.8 because the indirect addressing mode would make the algorithm appear more convoluted and unnecessarily obfuscate the basic idea. More information on this mode appears in appendix A.1.

In Fig. 5.9 the basic DIF0-R4B used in the 64 complex point FFT is shown. It incorporates two changes as compared to Fig.

5.8. First of all the indirect addressing mode is used. It is assumed that the two auxiliary registers have been initialized to 8AH and 8DH to point them at the second page, and ARP = 0. The 'H' signifies a hexadecimal number.

The temporary addressing has been contrived such that on completing the R4B the auxiliary registers are left pointing at their original locations. Consequently the initialization of the auxiliary registers need take place only once at the start of the FFT. The second change is the elimination of the temporary location Temp4. The code of Fig. 5.8 has been arranged to make this possibility obvious. Temp4 is the last calculated of the temporary locations. Section 2 could be followed by either section 3 or section 4 and still allow a 34 instruction R4B. But by choosing section 3 as in Fig. 5.8 the need to store Temp4 is eliminated. This is because the section which follows its calculation is also the section which uses it, hence the accumulator can be used as a temporary register for Temp4. The result of this change is that one more instruction may be saved and so Fig. 5.9 shows a 33 instruction DIF0-R4B. Since a 32 instruction version is not possible this is a reasonable result.

LAC	$x_{3r}, 15$		
ADD	$x_{1r}, 15$		
SACH	Temp1	$\text{Temp1} \leftarrow (x_{3r} + x_{1r})/2$	Section 1
SUBH	x_{1r}		
SACH	Temp2	$\text{Temp2} \leftarrow (x_{3r} - x_{1r})/2$	
LAC	$x_{1i}, 15$		
ADD	$x_{3i}, 15$		
SACH	Temp3	$\text{Temp3} \leftarrow (x_{1i} + x_{3i})/2$	Section 2
SUBH	x_{3i}		
SACH	Temp4	$\text{Temp4} \leftarrow (x_{1i} - x_{3i})/2$	
LAC	$x_{0r}, 14$		
SUB	$x_{2r}, 14$		
ADD	Temp4, 15		
SACH	x_{1r}	$x_{1r} \leftarrow x_{1r}/4$	Section 3
SUBH	Temp4		
SACH	x_{3r}	$x_{3r} \leftarrow x_{3r}/4$	
LAC	$x_{0i}, 14$		
SUB	$x_{2i}, 14$		
ADD	Temp2, 15		
SACH	x_{1i}	$x_{1i} \leftarrow x_{1i}/4$	Section 4
SUBH	Temp2		
SACH	x_{3i}	$x_{3i} \leftarrow x_{3i}/4$	
LAC	$x_{0i}, 14$		
ADD	$x_{2i}, 14$		
ADD	Temp3, 15		
SACH	x_{0i}	$x_{0i} \leftarrow x_{0i}/4$	Section 5
SUBH	Temp3		
SACH	x_{2i}	$x_{2i} \leftarrow x_{2i}/4$	
LAC	$x_{0r}, 14$		
ADD	$x_{2r}, 14$		
ADD	Temp1, 15		
SACH	x_{0r}	$x_{0r} \leftarrow x_{0r}/4$	Section 6
SUBH	Temp1		
SACH	x_{2r}	$x_{2r} \leftarrow x_{2r}/4$	

Fig. 5.8 A 34 instruction DIF0-R4B

LAC	$x_{3r}, 15$	
ADD	$x_{1r}, 15$	
SACH	$*_{1r}, 0$	$8AH \leftarrow (x_{3r} + x_{1r})/2$
SUBH	x_{1r}	
SACH	$*_{1r}, 1$	$8BH \leftarrow (x_{3r} - x_{1r})/2$
LAC	$x_{1i}, 14$	
ADD	$x_{3i}, 14$	
SACH	$*_{1i}, 0$	$8DH \leftarrow (x_{1i} + x_{3i})/2$
SUB	$x_{3i}, 15$	$ACC \leftarrow (x_{1i} - x_{3i})/2^{14}$
ADD	$x_{0r}, 14$	
SUB	$x_{2r}, 14$	
SACH	x_{1r}	$x_{1r} \leftarrow x_{1r}/4$
SUB	$x_{1i}, 15$	
ADD	$x_{3i}, 15$	
SACH	x_{3r}	$x_{3r} \leftarrow x_{3r}/4$
LAC	$x_{0i}, 14$	
SUB	$x_{2i}, 14$	
ADD	$*_{15}, 0$	
SACH	x_{1i}	$x_{1i} \leftarrow x_{1i}/4$
SUBH	$*_{1i}, 1$	
SACH	x_{3i}	$x_{3i} \leftarrow x_{3i}/4$
LAC	$x_{0i}, 14$	
ADD	$x_{2i}, 14$	
ADD	$*_{15}, 1$	
SACH	x_{0i}	$x_{0i} \leftarrow x_{0i}/4$
SUBH	$*_{0i}, 0$	
SACH	x_{2i}	$x_{2i} \leftarrow x_{2i}/4$
LAC	$x_{0r}, 14$	
ADD	$x_{2r}, 14$	
ADD	$*_{15}$	
SACH	x_{0r}	$x_{0r} \leftarrow x_{0r}/4$
SUBH	$*_{0r}$	
SACH	x_{2r}	$x_{2r} \leftarrow x_{2r}/4$

Fig. 5.9 The basic 33 instruction DIF0-R4B

5.5 The DIF2-R4B

The DIF2-R4B must perform the operations of a DIF0-R4B and two nontrivial complex multiplications as well. All of the DIF2-R4B's are identical to the R4B shown below.

This R4B is a special case in that all the complex coefficients have properties that allow for very economical coding. The w^0 and w^{16} coefficients are trivial. Multiplication by $w^0 = 1$ requires no operations at all. The other trivial coefficient is $w^{16} = -j$. It is seen from $(a+jb)(-j) = (b-ja)$ that no multiplier is needed. The operation x_2w^{16} is equivalent to interchanging the locations of the real and imaginary components and then complementing the new imaginary component.

Complementing a data memory location requires 3 instructions. Clearly it is not strictly necessary to include this complementing sequence and so a DIF2-R4B which includes it is referred to as the long version. The faster alternative is to

leave out the sequence and perform an effective complementing action in the subsequent DIF-R4B which addresses X_{2i} . For example, where the instruction ADD X_{2i} appears it is replaced by SUB X_{2i} and vice versa. This manoeuvre is applicable in each of the 5 instances of where a DIF2-R4B occurs saving $(5)(3) = 15$ instructions total with the short form.

In general a complex multiplication has 4 real multiplications and 2 additions, requiring 13 instructions on the TMS32010. An example coding is given below:

$$(X_r + jX_i)(W_r + jW_i) = (X_r W_r - X_i W_i) + j(X_r W_i + X_i W_r) \quad (5.29)$$

$$(X) \quad (W) \quad = \quad P_r \quad + \quad jP_i$$

```

LT      Xr
MPYK    Wr
PAC
LT      Xi
MPYK    Wi
SPAC    -1
SACH    Pr, 4
MPYK    Wr
PAC
LT      Xr
MPYK    Wi
APAC
SACH    Pi, 4

```

$P_r \leftarrow X_r W_r - X_i W_i$
 $P_i \leftarrow X_r W_i + X_i W_r$

A general complex multiply.
13 instructions total.

A new instruction has been introduced, SPAC, which means subtract the P register from the accumulator. The code has been arranged to use T as a temporary register for X_i saving one LT

instruction, but without constraints on W_r and W_i this is the best that can be done.

The special case of W^8 and W^{24} is different. In both the coefficients the magnitudes of the real and imaginary components are equal. Only two real products need be formed instead of four, saving 3 instructions over the general case.

$$(X_r + jX_i)(W_s - jW_s) = (X_r W_s + X_i W_s) + j(X_i W_s - X_r W_s) \quad (5.30)$$

$$(X) \cdot (W^8) = P_r + jP_i$$

LT X_i
 MPYK W_s^i
 PAC
 LT X_r
 MPYK W_s^r
 SPAC
 SACH $P_i, 4$
 APAC
 APAC
 SACH $P_r, 4$

Complex multiply by $W_s - jW_s$.
 10 instructions total.

The above example uses both the accumulator and the P register as temporaries. The code was arranged to have an APAC as the next to last instruction which forced calculation of P_i before P_r . This latter is of no consequence but positioning of the APAC is. Since one more complex multiplication by W^{24} is required, the last APAC may be absorbed into the first LT instruction of the following W^{24} multiplication by substituting

an LTA. Thus the total code for the two complex multiplications is 19 instructions instead of 20.

The presence of the two complex multiplications in the DIF2-R4B eases coding restrictions. With the DIF0-R4B it was necessary to store all the X outputs directly on the first page. A different procedure is possible with the DIF2-R4B. The X_1 and X_3 values can be stored in temporary locations on the second page. This is useful because it avoids overwriting the input values allowing a 32 instruction core. The core is that part of the R4B which does not include the multiplications. After the core has been completed the multiplication segments can access the temporaries for X_1 and X_3 . By storing the products back on the first page the DIF2-R4B is computed effectively in-place without any overhead from the temporary storage. With these ideas in mind it is now possible to do the coding.

Since all the major aspects have already been discussed they will not be repeated again in a detailed description of the code development. Instead, it is simply stated that the result is a 51 instruction DIF2-R4B and an example listing is shown in Fig. 5.10.

5.6 The DIF3-R4B and the DIF4-R4B

The DIF3-R4B and the DIF4-R4B both use the same 32 instruction core shown in Fig. 5.10 for the DIF2-R4B. This core is followed by code for 3 nontrivial complex multiplications to code a DIF3-R4B and 4 nontrivial complex multiplications to code a DIF4-R4B. The simplest coding makes use of the 13 instruction code segment of a general (no special properties) complex multiply. As discussed previously, when these segments are concatenated for multiple multiplies it is only the last segment which must use the full 13 instructions. The preceding segments require only 12 instructions. When coded in this way, the result is 69 instructions for the DIF3-R4B and 81 instructions for the DIF4-R4B. Of course, there is no reason to be so general. A closer look at all the DIF3-R4B's shows that in every case there is exactly one multiplication by either w^8 or w^{24} . As explained in section 5.5 these may be coded with 3 instructions less than a completely general multiplication. Consequently a DIF3-R4B actually requires only 66 instructions.

It is characteristic of an in-place FFT algorithm based on a Cooley-Tukey type decimation to shuffle the order of either the input or output data sequence. The particular pattern of the shuffling depends on the number of decimations and the

Fig. 5.10 The DIF2-R4B

Basic 32 instruction core

Assume: AR0 points 8AH

AR1 points 8DH

ARP = 0

LAC	$x_{0r}, 14$	
SUB	$x_{2r}, 14$	
ADD	$x_{1i}, 14$	
SUB	$x_{3i}, 14$	
SACH	$*+r$	$8AH \leftarrow X_{1r}/4$ and AR0 points 8BH
SUB	$x_{1i}, 15$	
ADD	$x_{3i}, 15$	
SACH	$*+0, 1$	$8BH \leftarrow X_{3r}/4$
LAC	$x_{0i}, 14$	
SUB	$x_{2i}, 14$	
ADD	$x_{3r}, 14$	
SUB	$x_{1r}, 14$	
SACH	$*+r$	$8DH \leftarrow X_{1i}/4$ and AR1 points 8EH
SUB	$x_{3r}, 15$	
ADD	$x_{1r}, 15$	
SACH	$*+r$	$8EH \leftarrow X_{3i}/4$ and AR1 points 8FH
LAC	$x_{0r}, 14$	
ADD	$x_{2r}, 14$	
ADD	$x_{3r}, 14$	
ADD	$x_{1r}, 14$	
SACH	x_{0r}	$x_{0r} \leftarrow X_{0r}/4$
SUB	$x_{3r}, 15$	
SUB	$x_{1r}, 15$	
SACH	$*+r$	$8FH \leftarrow -X_{2i}/4$
LAC	$x_{0i}, 14$	
ADD	$x_{2i}, 14$	
ADD	$x_{1i}, 14$	
ADD	$x_{3i}, 14$	
SACH	x_{0i}	$x_{0i} \leftarrow X_{0i}/4$
SUB	$x_{1i}, 15$	
SUB	$x_{3i}, 15$	
SACH	x_{2r}	$x_{2r} \leftarrow (X_2 W^{16})_r$

continued next page..

DIF2-R4B continued.

Complex multiplications.

ZAC		complementing the new X_{2i}	} long version only
SUB	*-,0,0	AR1 points 8EH	
SACL	x_{2i}	$x_{2i} \leftarrow (X_2 W^{16})_i$	

*multiply X_3 by W^{24}

LT	*-,1	$T \leftarrow X_{3r}$ and AR0 points 8AH
----	------	--

MPYK -2896

PAC

LT	*-	$T \leftarrow X_{3i}$ and AR1 points 8DH
----	----	--

MPYK -2896

SPAC

SACH	$x_{3r,4}$	$x_{3r} \leftarrow (X_3 W^{24})_r$
------	------------	------------------------------------

APAC

LTA	*,0	$T \leftarrow X_{1i}$
-----	-----	-----------------------

SACH	$x_{3i,4}$	$x_{3i} \leftarrow (X_3 W^{24})_i$
------	------------	------------------------------------

*multiply X_1 by W^8

MPYK 2896

PAC

LT	*	$T \leftarrow X_{1r}$
----	---	-----------------------

MPYK 2896

SPAC

SACH	$x_{1i,4}$	$x_{1i} \leftarrow (X_1 W^8)_i$
------	------------	---------------------------------

APAC

APAC

SACH	$x_{1r,4}$	$x_{1r} \leftarrow (X_1 W^8)_r$
------	------------	---------------------------------

long version = 54 instructions

short version = 51 instructions

radix's of the decimations.

5.7 Ordering the input sequence

The ordering of the output for the algorithm described here is the natural sequence. Referring only to the indexes this is represented by:

0,1,2,3,... 62,63 (s0)

This is the sequence of numbers appearing at the extreme right of flow diagram Fig. 5.1.

The ordering of the input sequence is implicit in the equations of section 5.2.1. There are two radix 4 decimations which take place. In the first, eq. (5.5) breaks the 64 point transform into 4 sixteen point transforms. The input sequence of the first 16 point transform is every 4th value of the natural sequence beginning with index 0. The second transform takes every 4th value beginning with index 1 and so on. The input sequences for the four transforms are then:

$G_0(k): 0, 4, 8, 12, 16, 20, \dots$	60	(s1)
$G_1(k): 1, 5, 9, 13, 17, 21, \dots$	61	(s2)
$G_2(k): 2, 6, 10, 14, 18, 22, \dots$	62	(s3)
$G_3(k): 3, 7, 11, 15, 19, 23, \dots$	63	(s4)

where the $G(k)$ are the four 16 point transforms.

The second decimation takes each $G(k)$ and breaks it down into 4 transforms of 4 points. This is given in eq. 5.15. The effect on ordering is as with the first decimation. For example, designate 4 transforms which compose $G_0(k)$ by $H_{00}(k)$, $H_{01}(k)$, $H_{02}(k)$ and $H_{03}(k)$. The input sequence for $H_{00}(k)$ is every 4th value from s1 beginning with index 0. The sequence for $H_{01}(k)$ is every 4th value from s1 beginning with index 4 and so on.

The resulting sequences are:

$H_{00}(k): 0, 16, 32, 48$	(s5)
$H_{01}(k): 4, 20, 36, 52$	(s6)
$H_{02}(k): 8, 24, 40, 56$	(s7)
$H_{03}(k): 12, 28, 44, 60$	(s8)

Similarly the input sequences for the remaining $G_i(k)$ are designated by $H_{iy}(k)$:

$H_{10}(k)$: 1,17,33,49	(s9)
$H_{11}(k)$: 5,21,37,53	(s10)
$H_{12}(k)$: 9,25,41,57	(s11)
$H_{13}(k)$: 13,29,45,61	(s12)
$H_{20}(k)$: 2,18,34,50	(s13)
$H_{21}(k)$: 6,22,38,54	(s14)
$H_{22}(k)$: 10,26,42,58	(s15)
$H_{23}(k)$: 14,30,46,62	(s16)
$H_{30}(k)$: 3,19,35,51	(s17)
$H_{31}(k)$: 7,23,39,55	(s18)
$H_{32}(k)$: 11,27,43,59	(s19)
$H_{33}(k)$: 15,31,47,63	(s20)

The resulting input sequence for the complete transform is the concatenation of s5 through s20 inclusive. These indexes are the numbers which appear at the extreme left of flow diagram Fig. 5.1. It is often desirable to have the FFT accept the data in natural sequence in which case an appropriate shuffling routine must be coded. Comparing the input sequence with the natural sequence shows that 48 values do not match up. A closer look shows this is the result of 24 interchanges. Each value has a

real and imaginary part and so 48 real interchanges can form the input data sequence from the natural sequence. One real interchange requires 4 instructions as shown below. This is the best that can be done.

Interchange a and b:

LAR	AR0,a	
LAR	AR1,b	
SAR	AR0,b	
SAR	AR1,a	a and b are interchanged.

The complete input shuffling routine is $(48)(4) = 192$ instructions or 38.4 usec.

At this point it is important to note that this input shuffling routine is presented only for completeness in the general case. In coding the spectrograph software package it is possible to circumvent this routine without introducing any additional overhead elsewhere, thus saving 38.4 usec. The method is trivial and is explained in chapter 6.

5.8 The doubling algorithm

An algorithm for finding the $2N$ long transform of an N long real data sequence is given in [20]. The idea hinges on the fact that the DFT of a real sequence is conjugate even:

$$X(k) = X^*(-k) \quad (5.31)$$

where $X(k)$ is the DFT of $x(n)$, the $*$ operator indicates a complex conjugate, and $X(-k) = X(N-k)$.

This is readily shown by writing the real sequence $x(n)$ as $0.5[x(n) + x^*(n)]$, then:

$$X(k) = 0.5 \sum_{n=0}^{N-1} x(n) \exp(-j2\pi kn/N) + 0.5 \sum_{n=0}^{N-1} x^*(n) \exp(-j2\pi kn/N) \quad (5.32)$$

$$X(k) = 0.5[X(k) + X^*(-k)] \quad (5.33)$$

Substituting eq. 5.33 into eq. 5.31 shows the equality is true.

The method is to take the $2N$ long real sequence $y(n)$ and separate it into $2N$ long real sequences $x_1(n)$ and $x_2(n)$ as follows:

$$x_1(n) = y(2n) \quad (5.34)$$

$$x_2(n) = y(2n+1)$$

$$\text{where } n = [0, N-1]$$

The $x_1(n)$ and $x_2(n)$ are used to form the N long complex sequence $x_c(n)$ which is the input sequence for the N -long FFT.

$$x_c(n) = x_1(n) + jx_2(n) \quad (5.35)$$

From the linearity of the DFT:

$$X_c(k) = X_1(k) + jX_2(k) \quad (5.36)$$

Replacing k with $-k$ and complementing both sides of eq. 5.36:

$$X_c^*(-k) = X_1^*(-k) + jX_2^*(-k) \quad (5.37)$$

From eq. 5.31, $X_1^*(-k) = X_1(k)$ and $X_2^*(-k) = X_2(k)$:

$$X_c^*(-k) = X_1(k) + jX_2(k) \quad (5.38)$$

Equations 5.36 and 5.38 are solved to extract $X_1(k)$ and $X_2(k)$:

$$X_1(k) = 0.5[X_C(k) + X_C^*(-k)] \quad (5.39)$$

$$X_2(k) = 0.5j[X_C^*(-k) - X_C(k)] \quad (5.40)$$

Note from eqs. 5.34 that $X_1(k)$ and $X_2(k)$ are the N long DFT's of the even and odd indexed values of the $2N$ long sequence $y(n)$. Thus $X_1(k)$ and $X_2(k)$ are the transforms which result in the first step of a radix 2 decimation-in-time algorithm for $Y(k)$. They are combined to form $Y(k)$ in the manner of that algorithm:

$$Y(k) = X_1(k) + X_2(k)W_{2N}^{-k} \quad (5.41)$$

$$\text{where } W_{2N}^{-k} = \exp(j2\pi kn/2N), \quad k = [0, N-1]$$

Equation 5.41 is for only the first N values of the $2N$ long $Y(k)$ but since $Y(k)$ is conjugate even it is completely defined.

5.9 Benchmark comparison

As indicated in the introduction to this chapter benchmarks exist for the calculation of a 64 point complex transform on the TMS32010 processor. With the FFT software now fully developed it is possible to determine the execution time. The basis for comparison will include the time required for the input ordering routine, even though this is not needed in the spectrograph version.

The calculation is:

Type of R4B	Number used	Number of insts.	Product
DIF0	24	33	792
DIF2	5	51	255
DIF3	10	66	660
DIF4	9	81	<u>729</u>
subtotal =			2436 insts.

Every instruction used in the coding executes in 0.2 usec. In the worst case, it is necessary to initialize the data page, the auxiliary register pointer, and both auxiliary registers every time the FFT is executed. This requires 0.8 usec, and 38.4 usec are added for the input ordering.

The total is:

$$(2436 \text{ inst.})(0.2 \text{ usec/inst.}) + 38.4 \text{ usec} + 0.8 \text{ usec} = 526.4 \text{ usec.}$$

This compares favorably with the fastest known benchmarks as indicated in the following table:

Fig. 5.11 Benchmark comparison

<u>Source</u>	<u>Time</u>
Texas Instruments [18]	783 usec
D.S.P.S. [19]	535 usec
This work	526.4 usec

It is difficult to reduce the execution time much further since every reduction by 1 usec means eliminating 5 instructions.

Chapter 6

Overview of the speech spectrograph

6.1 Introduction

A novel speech spectrograph system has been designed in accordance with the material developed in the preceding chapters. This chapter presents an overview of the system and outlines the means of integrating its components.

Section 6.2 presents the basic hardware configuration of the system. Section 6.3 discusses the allocation of software tasks between two processors and section 6.4 discusses quite briefly the DSP software modules required for the complete spectrograph calculation.

Section 6.5 discusses the DSP hardware module which integrates the TMS32010 processor with an IBM compatible microcomputer. It is not the intent of this section to present the methodology or rationale for the design of the DSP module. Such a presentation would necessarily involve a detailed look at the 8088 processor architecture and the internal structure of the

microcomputer, which is a topic in itself. Therefore the discussion is restricted to a high-level block diagram description and a few comments on the key constraints.

Section 6.6 presents some alternate configurations for the spectrograph system.

6.2 Basic configuration

The spectrograph system is illustrated in Fig. 6.1. The main component of the system is the microcomputer with the DSP module inserted into one of the general purpose expansion slots. The DSP module although internal to the microcomputer is visible because of the cut-away view.

A 25 conductor ribbon cable connects the main board of the DSP module to its subcomponent, the analog interface. The analog interface is external to the computer and has the microphone plugged into it.

Two displays are shown. One is a text-only monochrome monitor used to display menus and prompts to the user. The other is a composite monochrome monitor reserved for the 16 level

Fig. 6.1 A novel speech spectrograph system.

spectrograph image. The keyboard provides means for user input.

6.3 Task allocation

The spectrograph is a two-processor system. The TMS32010 is dedicated to the DSP operations which include all operations performed on a 128 point speech segment to yield a 128 point spectral cross-section. These are described further in section 6.4. All other tasks such as obtaining speech samples, displaying menus and prompts, responding to the keyboard and filling the display RAM are considered control functions and are assigned to the Intel 8088 processor. The 8088 is the processor of the microcomputer host.

6.4 DSP software

A block diagram of the DSP software is shown in Fig. 6.2. There are 7 distinct code modules under the control of an 8th, TMSCONTROL. TMSCONTROL is a supervisory routine which controls the calling of the other modules based on orders from the 8088 processor. In this way bus contention is avoided and considerable flexibility is provided in the DSP software. The

Fig. 6.2 DSP software for the speech spectrograph system.

main application for controlled calling is in partial processing, where for example it may be desired to perform an FFT on some data without the other functions. Of course, the input module must always be called to obtain new data from the 8088 and the output module is always needed to send back the results.

6.4.1 Input

The purpose of the input routine is to load the data memory of the TMS32010 with a 128 point speech segment passed from the 8088 processor through TMSBUF. TMSBUF is described in section 6.5. The main idea of the input routine is that it loads up the data RAM according to the requirements of the FFT and doubling algorithm discussed in chapter 5. This means the even and odd indexed values of the input data are sent to the locations for the real and imaginary components of the FFT, and furthermore they are loaded according to the radix 4 shuffled pattern discussed in section 5.7. This special ordering takes no longer than a natural sequence and so the need for a distinct input ordering routine for the FFT is eliminated without overhead. Thus the FFT used in the spectrograph system is 38.4 usec faster than an FFT for the general case needing natural ordering.

6.4.2 Hanning window

The Hanning window routine implements the operation of eq. 4.20 in chapter 4. It is a slightly convoluted coding since it must consider that the input routine presents the data values in the radix 4 shuffled pattern required by the FFT and doubling algorithm.

6.4.3 FFT and doubling algorithm

The FFT is fully discussed in chapter 5. Limitations in the available program memory space force the use of looped code structures in the doubling algorithm of section 5.8. Thus the auxiliary registers are used to address the FFT outputs and it is very important that the outputs appear in natural order, or else time consuming index calculations would be required. This point indicates that the FFT structure having a radix 4 shuffled input and normally ordered output is faster in the spectrograph application than the alternate structure with a normally ordered input and radix 4 shuffled output. Executed alone, either FFT may be equally fast, but when integrated into the spectrograph system the latter cannot be made as fast overall.

6.4.4 Magnitude estimation

The output of the doubling algorithm is a sequence of complex numbers. Determining their magnitudes exactly is very time consuming because of the iterative square root routine required. The problem can also be reformulated in terms of trigonometric functions and division routines, but this is also a slow process. Efficiency is important because 65 magnitudes are calculated.

The solution used in this work was to compromise accuracy by calculating an estimate of each magnitude using an algorithm from [21]. The idea is to form a linear combination of the real and imaginary components to obtain the estimate. No square root or division is required and by the careful use of shifting techniques even multiplication is avoided.

It is of course necessary to determine which of the real or imaginary components is the larger before forming the linear combination. The specific algorithm chosen for the spectrograph system has a mean error of 4 % and a peak error of 6 %. This is insignificant considering the massive compression in dynamic range which occurs in the following module.

6.4.5 Dynamic range compression

The largest number which could be output from the magnitude estimation routine is 15 bits in length. The minimum value is 0. This large dynamic range must be reduced to 16 levels or 4 bits because of the limited gray level capability of the display device.

A natural choice might be to calculate the base 2 logarithm which performs the mapping quite conveniently and can be approximated very efficiently using a variation on the method of [22]. However a simple logarithmic compression has the unfortunate characteristic of too heavily weighting small amplitudes relative to the larger amplitudes. This in turn has the visible effect of enhancing very low level background noise and obscuring the spectrograph display. The solution devised for this work is utterly simple, executes as quickly, and produces much better images than the base 2 logarithmic method.

The idea is merely to define 16 bins, each of which is assigned a 4 bit code. Each of the bins is assigned two boundary values, but adjacent bins share one boundary and so only 15 boundary values are needed. The magnitude differences between boundary values are unconstrained except that the boundaries must

be arranged in ascending order. To compress a number, its magnitude is compared to sequential bin boundaries until the first bin with a boundary greater than the test magnitude is determined. The 4 bit bin number is the result of the compression. Contrast and intensity control of the spectrograph image is achieved by varying the boundary values.

Although not restricted to logarithms, it is interesting that this algorithm can be used to approximate logarithm functions of arbitrary base.

6.4.6 Interpolation

The interpolation is an ordinary linear interpolation applied to the 65 outputs of the dynamic range compression module to get 128 outputs. It is used to enlarge the vertical dimension of the display for esthetic reasons. An alternate approach that is sometimes used in other applications is to output each point twice, avoiding any calculation. This was experimented with but it was found that little blocks were discernable within the spectrograph and so the linear interpolation is used.

For example the use of two monitors is an optional convenience. One monitor has ample capacity to display both text and a spectrograph, two monitors simply allows more text and a larger spectrograph.

Additionally, Fig. 6.1 may imply that all speech analyzed by the system must originate from its own analog interface. This is not the case. Sound files originating from any source may be transferred to the system by either diskettes or through an RS-232 interface. In fact during development the analog interface was the last phase to be completed. Speech files to be analyzed were obtained from a large VAX-based ILS (interactive laboratory systems) system through the RS-232 port of the microcomputer.

DSP software, the DSP hardware is not optimized for speed. Nevertheless it increases processing speed approximately 50 times, well over one order of magnitude improvement. Some considerations in the design of the module were to have it useable with as many IBM compatible microcomputers as possible, to have its existence completely transparent to the microcomputer, and to have a very nominal component cost.

There are two basic methods by which expansion circuitry might communicate with the 8088 processor. These are through its I/O address space or memory address space. Despite the sheer simplicity and utility of the memory mapped address scheme it is not used by the DSP module because of the compatibility and transparency constraints. Many microcomputers and commercial software packages make full use of all available memory space and so their function is corrupted if even one byte is seized for another purpose.

With the I/O mapped addressing scheme this problem may be circumvented. It is true that commercial expansion hardware frequently uses I/O address space, but contention is avoidable by adhering to established standards. In the IBM PC and PC-XT technical reference manual large blocks of I/O addresses are reserved for the common functions such as video displays, and a

few are unallocated. Eight of the latter are sufficient for all communication between the TMS32010 and 8088 processors.

A block diagram of the circuitry for obtaining analog signals is shown in Fig. 6.3. It is a fairly standard method of bringing microphone level signals up to the amplitude required for A/D conversion. The preemphasis circuit inserts a first-order pole at 500 hz to render the higher speech frequencies more visible.

The main circuit card of the DSP module is shown in Fig. 6.4, in block diagram form. A ribbon cable carries the digitized speech from the analog interface to the buffer at the lower right of the diagram. This buffer provides an interface to the 8088 expansion bus. The 8088 processor stores speech files in the microcomputer memory, moving segments of a file to TMSBUF for processing when required. In this way it is possible to work with massive speech files and yet keep the memory size of TMSBUF down to only 128 words, the length of one speech segment. TMSBUF is a 128 word long memory interface buffer between the two processors. It is used to reconcile the vastly different working speeds of the processors and the fact that while the TMS32010 accesses data words, the 8088 only accesses bytes. Thus TMSBUF is divided into MSB and LSB portions as shown at the upper right of Fig. 6.4.

Fig. 6.3 Analog interface.

Fig. 6.4 DSP module card.

In transferring 128 speech samples to TMSBUF it was considered too wasteful to assign 128 I/O addresses and so an automatic addressing circuit was designed. This A.A.C. reduces the I/O address space requirement to two, one for the MSB and one for the LSB of the speech samples. It basically functions on a stack addressing principle with TMSBUF. It is a highly flexible design providing bidirectional access to TMSBUF by either processor with autoincrement and autodecrement modes by bytes and/or words. All of these functions are required by the spectrograph software and are selected as required by writing to the appropriate status latch, E8STAT or TMSSTAT.

These status latches are used by the processors to inform each other of what the other is doing. As each task is completed their status is updated to reflect the new situation. This is very helpful in coordinating the asynchronous movement of data between processors, memory, and buffers. In this way any coding changes which alter the execution time of a software module are automatically compensated for.

Additionally, some bits of the status registers are hardwired to control logic which prevents any contention for the internal 8/16 bit data bus of the DSP module. Such a situation could only arise due to a significant coding error but since humans are

involved, the precaution is necessary.

The remaining circuitry on the module consists of a programmable sample clock and program memory for the TMS32010.

The sample clock is provided for obtaining speech samples, the one operation which must not be asynchronous. The maximum sampling frequency is 22 khz, the particular rate is chosen by the 8088 writing to the SAMPLE latch.

The 4kword program memory of the TMS32010 is organized as 4k by 8 PROMs. This is the most inexpensive storage and was made possible by avoiding any write operations to program memory in the DSP software. The software actually occupies all but 15 locations of the 4kword space.

6.6 Alternate configurations

The diagram of Fig. 6.1 is the current configuration of the system but is only one possibility.

For example the use of two monitors is an optional convenience. One monitor has ample capacity to display both text and a spectrograph, two monitors simply allows more text and a larger spectrograph.

Additionally, Fig. 6.1 may imply that all speech analyzed by the system must originate from its own analog interface. This is not the case. Sound files originating from any source may be transferred to the system by either diskettes or through an RS-232 interface. In fact during development the analog interface was the last phase to be completed. Speech files to be analyzed were obtained from a large VAX-based ILS (interactive laboratory systems) system through the RS-232 port of the microcomputer.

Chapter 7

Conclusion

7.1 Introduction

There has been a significant body of work done in developing the speech spectrograph system which is not directly discussed in this thesis. This is to keep it from becoming overly long and detracting from its focus. Thus the current status of the system might seem somewhat advanced. Actually the speech spectrograph system was completed in January 1985. It could accept data from diskettes, an RS232 port or its own analog interface and included a small library of Fortran callable 8088 assembler routines. These routines were to allow the rapid development of user-friendly features by permitting highlevel language programming. At that time the system prompts were particularly terse and the only function was producing spectrographs.

Section 7.2 describes the current (March 1985) status of the system and indicates future plans. Section 7.3 presents some conclusions drawn from the work.

7.2 Current status

Since January 1985 the high level language features of the spectrograph system software have been further developed at the Communications Research Center of Canada (C.R.C.). In particular Dr. S. Shlien has added additional features and made the system quite user friendly. The system now functions interactively. Default values for key parameters such as sampling rate, contrast and brightness of the spectrograph image appear on-screen in menu form. The user may choose the default values or select new values with simple cursor controls. A Help file explains the systems options which now include display of time waveforms and storage of spectrographs on diskette files.

It takes under 6 seconds to complete a spectrograph image for a 2 second speech file. Of this time, only 0.9 seconds are spent with DSP code on the TMS32010.

The images produced include calibration information for the time and frequency axis, and are comparable in quality to textbook examples.

Current work and future plans include patent applications, constructing copies of the system, developing an interactive

[21] Adams, Brady, 'Magnitude Approximations for Microprocessor Implementation', I.E.E.E. Micro, October, 1983

[22] Mitchell, 'Computer Multiplication and Division using Binary Logarithms', I.R.E. Trans. Electron. Comput., vol. EC-11, pp. 512-517, Aug. 1962

Fig. 7.1 Comparison to Kay Sonogram

Kay Sonogram

This work

7.4 Concluding remarks

The processing time of 6 seconds for a 2 second speech file is a reasonable figure which does not test the users patience. Since only 0.9 seconds are spent on DSP code, the 8088 processor is seen to be a bottleneck in moving data. DMA (direct memory access) techniques could make the system faster than real time if this were perceived as useful. However this is not recommended because the universality and low cost of the DSP hardware module would be compromised. The current module has tested successfully with several IBM compatible microcomputers. There is no known combination of compatible microcomputer and software package that is interfered with by the DSP module. Even manufacturer's diagnostics software with system configuration tests does not detect its presence.

Components cost for the DSP module is very modest, thus the goals of wide compatibility and economy have been met. The transparency of the DSP module to the host microcomputer means the latter need not be dedicated to the spectrograph system and can retain all previous capabilities.

In performing some experiments with the spectrograph system the usefulness of the quantizer described in section 6.4.5 became apparent. With thresholds set for base 2 logarithmic quantization consonants are generally not discernable but vowels show up well. For an example of this effect on trying to read a spectrograph, attempt to decipher the word containing the following vowels; e o and a. Clearly very little reading can be done with only vowels. Next consider these consonants only; s p c t r g p h. It is likely that the consonants provided better clues to the word 'spectrograph' than did the vowels. With flexible quantization consonants can be made much more visible and therefore the system utility is improved.

In another experiment 16 level color spectrographs were compared for readability to monochrome spectrographs. Color spectrographs were obtained by simply replacing the monochrome monitor with a color monitor. No other changes of any kind were required. The resulting display was a thermal mapping with black and red designating the lowest energies and blue and white the highest. The images were very interesting to see but of little use for reading. The color image renders the discreet amplitude

quantization inescapable. Identical sounds at different volumes produce images with different colors. Even vowel formants are difficult to follow. Monochrome images are different in that volume changes affect mainly the overall brightness, and the transitions between adjacent levels always look smooth. At least for the purpose of reading spectrographs, it seems that color is not a feature but rather a detriment.

Finally, the readability of the spectrographs from this system suggests the design approach was reasonable and the new benchmark reported in chapter 5 supports the methodology used to achieve it.

References

and

Appendix

References:

- [1] Ladefoged, 'A Course in Phonetics', Harcourt Bruce
Johanovitch Inc., New York, N.Y. 1975

- [2] G. Fant, 'The Acoustics of Speech', Proceedings of the Third
International Congress on Acoustics, 1959

- [3] W. Koenig, H.K. Dunn, and L.Y. Lacy, 'The Sound
Spectrograph', J. Acoust. Soc. Amer., vol. 18, pp 19-49, 1946

- [4] Potter, Kopp, Green, 'Visible Speech', Van Nostrand, N.Y.,
N.Y., 1947

- [5] Cooley, J.W., and Tukey, J.W., 'An algorithm for the machine
computation of complex Fourier Series', Math. Comput., vol. 19,
pp 297-301, Apr. 1965

- [6] Rothauser, E. and Maiwald, D., 'Digitalized sound
spectrography using FFT and multiprint techniques (abstract)',
J. Acoust. Soc. Am., vol. 45, p 308, 1969

[7] A.V. Oppenheim, 'Speech spectrograms using the fast Fourier transform', I.E.E.E. Spectrum, vol. 7, pp. 57-62, Aug 1970

[8] P. Mermelstein, 'Computer-Generated Spectrogram Displays for On-Line Speech Research', I.E.E.E. Trans. Aud. and Electroacous., vol. AU-19, pp. 44-47, Mar. 1971

[9] L.R. Morris, 'Fast Speech Spectrogram Reduction and Display on Minicomputer/Graphics Processors', I.E.E.E. Trans. Acoust. Speech and Sig. Proc., vol. ASSP-23, No. 3, June 1975

[10] F. Nolan, 'The phonetic bases of speaker recognition', Cambridge University Press, N.Y., N.Y., 1983

[11] Rabiner and Gold, 'Theory and Application of Digital Signal Processing', Prentice-Hall, Engelwood Cliffs, N.J., 1975

[12] Bergland, 'A guided tour of the fast Fourier transform', I.E.E.E. Spectrum, July 1969

[13] Harris, 'On the use of windows for harmonic analysis with the discrete Fourier transform', Proc. I.E.E.E, vol. 66, pp. 51-83.

[14] Nuttall, 'Some windows with very good sidelobe behavior',
I.E.E.E. Trans. ASSP, vol. ASSP-29, No. 1, Feb. 1981

[15] Oppenheim, Schafer, 'Digital Signal Processing', Prentice
Hall Inc., Englewood Cliffs, N.J.

[16] Schafer, Rabiner, 'Digital Representations of Speech
Signals', Proc. E.E.E.E., vol. 63, pp. 662-667, Apr. 1975

[17] Morris, 'Algorithm, Architecture, and Compiler Interaction
in DFT Programs for General Purpose Computers', I.E.E.E. Trans.
ASSP, to be published

[18] Electronic Design, Aug. 19, 1982, p154

[19] Electronic Design, Oct. 4, 1984, p15

[20] Cooley, Lewis, Welch, 'The fast Fourier transform algorithm:
programming considerations in the calculation of sine, cosine
and Laplace transforms', J. Sound Vibration, vol. 12, pp.
315-337, July 1970

[21] Adams, Brady, 'Magnitude Approximations for Microprocessor Implementation', I.E.E.E. Micro, October, 1983

[22] Mitchell, 'Computer Multiplication and Division using Binary Logarithms', I.R.E. Trans. Electron. Comput., vol. EC-11, pp. 512-517, Aug. 1962

Appendix A.1

Indirect addressing on the TMS32010

The TMS32010 has two auxiliary registers which may be used for indirect addressing. These are AR0 and AR1. Only one is active at a time, and just which is active is indicated by a one bit flag, ARP. If $ARP = 0$, AR0 is active. If $ARP = 1$, the AR1 is active.

Autoincrement and autodecrement by 1 are also possible. In either case the adjustment of the auxiliary register takes place after the data is accessed.

Use of the indirect addressing mode is indicated by the '*' symbol in the address field of the instruction. Autoincrement is indicated by '*' and autodecrement by '*-'.

If the instruction also has a shift field the desired shift is placed next in the address field of the instruction separated by a comma. For a shift off 14, possibilities are '*',14' or '*+',14' or '*-',14'.

It is also possible and sometimes necessary to set the ARP to change the active register for the next instruction. In this case, either a 0 or a 1 follows the shift field separated by a comma.

Several examples using the ADD instruction follow with explanations.

ADD * Take the contents of the memory location pointed to by the active auxiliary register and add them to the accumulator.

ADD *+ As above and also autoincrement the current

auxiliary register.

ADD *- As above but autodecrement.

ADD *-,14 As above but left shift data 14 bits

ADD *-,14,1 As above but make AR1 the next active
register.

ADD *+,8,0 Add with a left shift of 8 bits, autoincrement,
and make AR0 the next active register.

If the ARP field is left blank there is no change of active
auxiliary register.

If the shift field is blank there is no shift.

Appendix A.2

Code sizes and timings

It may be of interest to know the time and space requirements of the various spectrograph routines.

The total time required to produce a 600 spectral cross-section spectrograph is just under 6 seconds. The exact time varies by a few tenths of a second from measurement to measurement because some code paths are data dependent.

Of this 6 seconds, approximately 2 seconds are consumed by the control code of the 8088 processor and approximately 3 seconds are consumed by the display routine. No effort was made to optimize these routines for speed (user features took a higher priority during development) thus they may be made substantially faster. Both routines are on the order of 50 instructions long.

A precise breakdown of the time and space requirements of the TMS32010 code follows.

ROUTINE	MEMORY LOC's	INST. CYCLES	EXEC. TIME (usec's)
TMSControl	29	30	6
Input	130	130	26
Hanning Window	514	514	102
FFT 64 cmplx	2525	2525	505
Doubling alg.	473	1684	337
Magnitude est.	35	1039	208
Magnitude comp.	81	1836	367
Interpolate	20	578	116
Output	130	130	26
TOTALS ->	3937	8466	1693 usec's

Comments: The magnitude estimation and magnitude compression routines are data dependent and thus somewhat high representative values are given for the number of instruction cycles and execution times.

The FFT in the above table is the FFT developed for C.R.C. and used in the spectrograph, it is not the FFT discussed in chapter 5.

A 2 second (600 spectral cross-section) long spectrograph would need $1693 \text{ usec/s/section} \times 600 \text{ sections} = 1 \text{ second}$ of TMS32010 processor time.

END

22|06|88

FIN