WARNSIS II

A WARNING SIGNAL IDENTIFICATION SYSTEM

FOR THE HARD OF HEARING

Kim Dotto, P.Eng.

B.Sc. University of British Columbia, 1979
B.A.Sc. University of British Columbia, 1982

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

March 22 1995

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of ELECTRICAL ENGINEERING

The University of British Columbia
Vancouver, Canada

Date APRIL 24 1995

DE-6 (2/88)

Abstract

Having examined a number of common warning signals such a telephone rings, microwave oven alarms and continuous tone alarms, a software method, eliminating the need for specialized hardware, was developed for both recognizing known warning signals and learning new signals. The method, called WARNSIS II was first implemented on a PC-compatible computer using a Creative Labs signal acquisition board which provided a base for graphical signal analysis and for the development of algorithms. The algorithms were also implemented on the Texas Instruments TMS320C30 Evaluation Module (EVM) digital signal processing board to demonstrate that the algorithms can be easily ported to smaller, more cost effective platforms.

The WARNSIS II system operates in two modes: learning and recognition. In the learning mode a template for a signal is constructed by analysing the signal for spectral and temporal information. The frequency at which the maximum peak in the signal 's spectral energy occurs is determined and this value is saved in a template. The duration of the signal burst, as well as the interval time between signal bursts for repetitive signals, are determined and are also saved in the template. In the recognition mode, the spectrum of the incoming real-time signal is analysed to determine if any of its spectral components match those found in the template. If a match is found, timing information is then applied to rule out transients which would generate false alarms

The system has demonstrated the ability to recognize warning signals in high background noise environments and to correctly identify multiple overlapping warning signals.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AIC	-	Analog Interface Circuit
DAT	-	Dynamic Amplitude Threshold
DCT	-	Discrete Cosine Transform
DFT	-	Discrete Fourier Transform
DHT	-	Discrete Hartley Transform
DSP	-	Digital Signal Processor
DST	-	Discrete Sine Transform
DTW	-	Dynamic Time Warping
EVM	-	Evaluation Module
FFT	-	Fast Fourier Transform
FHT	-	Fast Hartley Transform
HMM	-	Hidden Markov Modelling
MBD	-	Minimum Burst Duration
MEM	-	Maximum Entropy Method
MIAT	-	Maximum Inter-Arrival Time
PC	-	Personal Computer
STAAA	-	Short Time Average Absolute Amplitude
STFT	-	Short Time Fourier Transform
VOC	-	Creative Voice File
WARNSIS	-	Warning Signal Identification System
WT	-	Wavelet Transform

Acknowledgement

I would like to thank my supervisor, Dr. C.A. Laszlo for his encouragement and advice during this project, and for his belief in my abilities from the start which allowed my to undertake this project. I would also like to thank my wife Patricia for her support and understanding and my children, Alexander and Laura for their unending diversion.

This project was funded by NSERC Operating Grant OGP0006701

Chapter 1

1.0 Introduction

1.1 Background

In everyday life we are constantly bombarded with information about our environment. Although humans are visually oriented, much of the important information, that which requires immediate attention, is transmitted by sound. Warning sounds such bells and buzzers are part of our everyday life both waking and asleep. Unlike visual information, which is not processed while asleep, auditory information is processed by the ear and brain continuously.

Approximately eight percent of the general population suffers from some level of hearing impairment. Approximately 70 out of every 1000 people are "hard of hearing", another 6 out of every 1000 are classified as late deafened adults, and 1 out of every 1000 are deaf [1,2]. Hard of hearing people are those who use speech as their main method of communication but have some impairment in their ability to hear. Such impairment may range from mild to profound. Late-deafened adults are those who have suddenly become deaf later in life usually due to accident, illness or medication related causes. This group still uses speech, but also must rely on speech reading or written aids such as transcription to understand what others say. Some late deafened adults also sign as a means of communication. People who are deaf from birth usually use signing as their main method of communication. Signing is a distinct language apart from any written or spoken language and promotes an independent culture just as the use of English or French does for speakers of those languages. Those who consider themselves to belong to this cultural group define themselves as Deaf. There are also Oral Deaf people, people who have been deaf from birth or an early age but have been taught to speak, rather than sign as their main method of communication. They may also rely on sign

occasionally but they rely on speech and speech reading as their main methods of communication.

Members of all groups are placed at various levels of disadvantage by their hearing loss, depending on the environment or situation. Such disadvantage can even be life-threatening, such as failing to hear a smoke detector, fire or burglar alarm. Even missing a phone call can have serious consequences. To help overcome this disadvantage many devices and approaches have been designed in the past to alert a Deaf or hard of hearing person to the presence of specific warning sounds. These fall into 3 main categories, threshold detectors, hard-wired or dedicated devices and hearing ear dogs[3].

1.1.1 Threshold Detectors

Most warning signals are designed to be heard above a general ambient noise level and, as such, are louder than most other sounds that one might encounter in a given environment. Threshold detectors function by exploiting this fact. They are designed to detect any increase in sound amplitude over a preset limit or set point. This can work reasonably well in an isolated environment where the device is required to identify a single warning sound. It can also be useful when multiple warning signals need to be monitored and all that is required is to alert the user that one of them has occurred without being specific as to which one.

One of the main drawbacks of threshold detectors, however, is that sounds which are louder than the preset limit but originate from sources other than the alarm or alert will also trigger the detector, giving a false positive indication. For example, a threshold detector set to detect a telephone ring may be erroneously triggered by a doorbell or a fire alarm or even the passage of a noisy truck.

Another disadvantage of a threshold detector is the need for adjustment when moving it from one location to another. If the detector were set to function correctly when in the same room as the telephone, the amplitude of the sound that reaches the detector if the user moves it to a different room may not be loud enough to trigger the detector. Adjusting the set point to a lower value may not be possible due to a greater ambient noise levels in the new location. The detector would then give a number of false positives that would render the device virtually useless.

Threshold detectors have been designed to operate on signal sources other than sound. For example, detectors for magnetic fields emitted by telephones exist, but the detectors must be located very near the telephone in order to pick up the relatively weak fields generated. This greatly reduces the range and effectiveness of such devices. Additionally, many newer telephones do not use electro-mechanical ringers and therefore generate no detectable magnetic fields.

1.1.2 Hard-wired / Dedicated Devices

Hard-wired or dedicated devices can overcome many of the problems found with the threshold detection devices described above. Hard-wired devices are electrically, and often mechanically, connected to the device they are designed to detect. The signalling device may also be hard-wired, although radio-operated remote pagers which use lights or vibration to signal the wearer are becoming popular [4]. For the telephone, rather than listening for an increase in ambient noise level, the detector is electrically connected to the telephone and monitors the incoming ringing signal from the telephone exchange. When the ringing signal is detected, the device signals the user by triggering the pager or flashing a lamp on and off.

There are three main disadvantages to this method:

i) a separate detector is required for each device that is being monitored;

ii) installation requirements can be prohibitive; and

iii) the devices lack portability.

The detectors for these types of devices are usually designed for a specific piece or type of equipment. For example, it is unlikely that a telephone detector would be suitable for connection to a door bell. In addition, many devices that a hard of hearing person would like to monitor, such as a microwave oven or smoke detector do not have commercially available detectors.

In those cases where a suitable specific detector is available, its installation may require expert assistance, or modification of the monitored equipment. This is both inconvenient and expensive for the user and can be hazardous should the installation be done incorrectly.

Lack of portability is a major drawback for the user with hard-wired devices. People are more mobile now than any time in the past and travel away from the home or office is much more common than ever before. For some jobs, especially at higher levels of responsibility, it is a major requirement of the job. Lack of mobility for the hard of hearing person can be a major impediment to career advancement. Business is more often conducted by phone and facsimile these days than by personal meeting. When travelling, the hard of hearing person will not be able to modify all, if any, of the devices that they would normally rely on at home with hard-wired detectors. Even a simple change of office or building could require significant time and expense to set up.

1.1.3 Hearing Ear Dogs

Like Seeing-Eye-Dogs, Hearing-Ear-Dogs are available for all hearing impaired people. These dogs are very useful, but they are also quite valuable and in short supply. It takes many months of training to prepare a dog and a reasonable "training" period for the recipient as well. In addition, dogs require regular "refresher" training. As training is a lengthy and difficult process, it is often not possible or practical to update the animals skills once the have been placed with the hearing impaired owner.

In addition, just as for Seeing-Eye-Dogs, not all hearing impaired people would be able to utilize such a dog. The hard of hearing person may not physically be able to care for a dog due to age or other disabilities or they may not wish to own a dog. A dog may not be suitable for other reasons such as living arrangements and there is increased cost and difficulties involved in the transportation of the dog when travelling.

Chapter 2

2.0 The WARNSIS Project

The objective of the WARNSIS (*WARN*ing *S*ignal *I*dentification *S*ystem) project is to develop a device that would overcome the problems and deficiencies noted with the devices and solutions described previously. As such, the device:

i) would be required to reliably recognize warning sounds for hard of hearing and deaf people in the presence of background noise, such as radio music, conversation and machinery;

ii) must be portable and not require any special connections to the warning signal generating sources;

iii) must have the ability to not only discriminate between different signals sources but must also have the ability to easily learn and recognize previously unknown signals.

In addition, it is an explicit long-term objective of the project to minimize the cost of manufacture in order to make the device accessible to as many users as possible.

There have been attempts previously to design devices with these capabilities in the past [5,6], but these devices have suffered from limitations in their abilities to learn a wide variety of signals or to discriminate between signals with similar timing and spectral characteristics.

Chapter 2 - The WARNSIS Project

2.1 WARNSIS I

To date there have been two phases of the WARNSIS project. In the initial phase of the WARNSIS project (WARNSIS I) much of the study was devoted to the characterization of the various types of warning signals in everyday life. WARNSIS I also produced an initial prototype device.

In WARNSIS I it was determined that the majority of signals that the device would encounter fell into two categories:

i) either single or repetitive burst signals (such as telephone rings);
ii) continuous signals (such as produced by fire alarms and smoke detectors).

It was also found that the warning signals examined in the study had at least one characteristic frequency or main harmonic above 900 Hz and below 5000 Hz [7].

Ambient noise was also characterized to be broad band with most of its spectra energy below a frequency of 300 Hz. This is common, as ambient noise in the usual office and home environments is generated by machinery such as fans and passing automobiles.

The frequency range between 300 Hz and 900 Hz was found to possess both noise and signal components. Most notably, warning signals from automobile horns (electric) and truck horns (air) fell into this category with their dominant frequencies in the 300 Hz to 500 Hz range. The human voice possesses energy at frequencies both inside and outside this range but most of the energy in the human voice is contained within this frequency range. The human voice rises about 5 dB from its base value in the range from 100 Hz to 600 Hz and then falls by approximately 6 dB, 9 dB, 12 dB and 15 dB in succeeding higher octaves [8].

The general recommendation for the design of warning devices is that such a device should be capable of producing an output sound level pressure (SPL) of at least 85 dB (with reference to 20 u Pa) at a distance of 10 ft from the device [9]. Additionally the warning signal should be at least 10 dBA SPL above the ambient noise level for warning devices in the home environment or 15 dBA SPL for those in public places [10]. This is a difficult condition to satisfy in real environments as most warning devices are preset at the time of manufacture or installation but the ambient noise levels where the device are used tend to vary with time.

2.1.1 WARNSIS I Recognition Scheme

The WARNSIS I recognition scheme relied on a two-part analysis [7]. The first part of the analysis was a timing analysis, the second was a spectral analysis. A signal was "recognized" only if both parts of the analysis were successful. In WARNSIS I, the timing analysis was performed first in order to speed processing. This was significant, because processing power was limited by the microprocessors available in the market at that time (1986-88).

The timing analyser included the following three processes:

i) the Dynamic Amplitude Threshold (DAT),
ii) the Minimum Burst Duration (MBD),
iii) the Maximum Inter-Arrival Time (MIAT).

The purpose of the DAT process was to identify a signal that may be of "interest" in order to start the recognition process. The DAT process continuously monitored the input signal level and updated a threshold level. If the input signal amplitude increased to twice the threshold level, a "signal of interest" was assumed to be present. The threshold was

continuously updated by averaging the previous threshold with the incoming signal amplitude. The threshold thus tracked an increase or decrease in the ambient noise level.

The MBD process was used to filter spurious burst noises before they reached the later recognition steps. The timing analyser monitored the incoming signal and measured the amount of time the input signal strength was above the threshold. If the MBD was less than a preset value the signal was ignored. If the MBD was longer than the preset limit a "valid" signal was assumed. If the MBD was longer than 4 seconds a continuous warning sound was assumed, if the signal was less than 4 seconds but longer than the preset minimum, a repetitive signal was assumed.

For repetitive signals the MIAT process was used to determine if the signal was in fact repetitive or a collection of random noise bursts of duration longer than the MBD. If two burst occurred in a time period less that the MIAT they were considered to belong to the same warning signal occurrence. If they occurred outside the MIAT they were considered to have been produced by separate warning events.

The timing analysis was followed by spectral analysis to determine if the identified burst was a valid signal. The spectral analyser was based on a NEC chip set designed to perform Dynamic Time Warping speech analysis and was implemented via an NEC μPD7761 and μPD7762 Pattern Matching Processor and Controller combination. The input signal was analysed by passing it through an eight frequency range filter bank and extracting the average energy contained in each of the eight frequency ranges between 100 Hz and 5000 Hz. The spectral analysis was then performed by comparing the output from these eight filters with pre-stored templates. If the output pattern from the filters matched one of the pre-stored templates, the signal was then considered to be validated and a warning to the user was issued. Similar filter bank approaches have been tried by other researchers as well [11].

2.1.2 Evaluation of WARNSIS I

2.1.2.1 Recognition Scheme

Although the two part system combining a timing analyser and spectral analyser seemed to give satisfactory results in low background noise cases, it produced a relatively higher number of false alarms in noisy environments and failed to detect a number of key signals. In fact, experience has shown that the failure of the device to recognize key signals in noisy environments was due to the sequence of the two-part timing and spectral analysis scheme .

In particular, as already noted, in WARNSIS I, the MBD detection process, used to determine if a burst was a warning signal or a transient, was applied before the spectral analysis to determine if the signal was worth examining. This meant that timing, not spectral energy content, was the main feature that the device detected. This, in combination with the MIAT measurement, caused the device to fail to detect repetitive signals in noisy environments. The original WARNSIS I device would identify a signal burst and then wait for the second in a series of supposed repetitive bursts. If a noise burst (or other warning signal) occurred during the waiting period, the original signal would be rejected because the MBD or MIAT did not match the template value. The spectral content of the intermediate burst was not examined, so there was no way for the device to differentiate if the intermediate burst was a true signal burst or if it was simply due to noise.

In an attempt to streamline processing, the original recognition scheme included three other compromises that made its real world use unsatisfactory. Firstly, the WARNSIS I device required at least one repetition to identify a repetitive signal. Since there is no guarantee in the real world that a repetitive signal will repeat, such signals should be identified on the basis of

a single appearance. For example, in an emergency situation the warning device could be damaged before giving the second warning burst or the phone, or the postman, "may only ring once". Thus, this recognition scheme would ignore the original burst since it detected no repetitions.

Secondly, the WARNSIS I recognition scheme based its initial signal determination on the increase in the average total signal amplitude (energy). This meant that from the time the beginning of a signal burst was identified, until the time that signal burst ended, the device could not recognize other warning signals. Thus, if a noise burst occurred and triggered the timing functions before a true signal burst occurred, the signal burst would be completely ignored, or added into the noise bursts timing. Also, if the device was monitoring a valid signal and another valid signal occurred, overlapping with the first, the second signal's timing characteristics would again be added to the first and neither signal would be successfully detected. This could quite easily occur in an office environment where a number of phones could ring simultaneously. Rather than giving a false positive, as would be the case for the threshold detector, the device would give no indication of any activity at all.

Lastly, although the DAT function successfully tracked the increase in ambient noise level to prevent false positives, this actually had the effect of "deafening" the device in high noise environments. For the start of a burst to be detected, an increase of twice the threshold limit was required. As the ambient noise level would rise, the threshold would rise. However, the absolute amplitude of the warning signal, such as the telephone ringer volume, would remain the same. Thus, the detection requirement of twice the threshold limit could not be met by the warning signal at high ambient noise levels.

2.1.2.2 Hardware Implementation

The WARNSIS I system consisted of approximately 80 components including 4 microprocessors and a number of specialty NEC spectral analysis chips. Given the level of complexity of the hardware, multiple processors, specialty chip sets, and the requirement of an additional PC to coordinate the overall function of the device, it would have been impossible to turn the prototype into a practical device due to reliability and price considerations.

However, the WARNSIS I project determined that it was possible to detect and recognize warning sounds in real time, and although the design of the device had several shortcomings, the WARNSIS I project provided the basis for the evaluation of the usefulness of the timing and spectral information in the detection of this class of signals.

2.2 WARNSIS II Objectives

Given the above analysis of WARNSIS I, the objectives of WARNSIS II are:

1. Redefine the detection and recognition algorithms to improve the performance in noisy environments and overcome the deficiencies in the original scheme;
2. Reduce the hardware requirements so that a viable, cost effective production device can be made.
3. Produce a system that is both easily learned and used by the end user
4. Produce a system that can be easily maintained and upgraded in the manufacturing environment.
5. Provide an overall system design, from both a hardware and software perspective, including support tools for further development and commercialization of the project.

Chapter 3

3.0 Sound and Signal Recognition Schemes

Currently most speech recognition systems are based on one of two basic mathematical models: Dynamic Time Warping (DTW) [12,13,14] or Hidden Markov Modelling (HMM)[12,13,15]. Both of these approaches are statistically based and are used because of the need to allow for the variations in speech that occur for the same word or words when spoken by various speakers or the same speaker at various times.

3.1 Speech Recognition Schemes

3.1.1 Dynamic Time Warping

Dynamic Time Warping is a useful technique when matching spectral patterns between a known signal (template) and a unknown incoming signal, which may contain timing differences. When a signal is " learned" and a template is created, the incoming signal is divided into smaller time segments. For each of these time segments a spectral analysis of the incoming signal is performed by a filter bank. The number of outputs from the filter bank depends on the spectral resolution required (For example: for WARNSIS I, this was eight outputs covering a frequency range between 100 Hz and 5000 Hz). The outputs from the filter bank give an estimation of the energy content in the spectral region covered by each of the filters in the bank. The outputs from these filters are stored in the templates for each time segment created during the learning process.

During the recognition process, the incoming signal is again divided into smaller time segments and the same spectral analysis is performed by the filter banks. For each time segment the output from the filter bank for the incoming signal is compared to the values stored

in the templates. If the output and template values match within a set of statistical parameters a match is indicated for that particular time segment.

The "Dynamic Time Warping" comes into effect by allowing for the misalignment of time segments. For example, if we wish to be able to recognize the word "hat", we would first analyse the voice of a speaker saying the word "hat". The length of time to say the word "hat" during the learning process may be 10 milliseconds and the learning process would divide this into 50 time segments. During the recognition phase, the speaker would again repeat the word "hat". However, because of changes in intonation and emphasis, which are normal during the speaking process, the new occurrence of the word hat may take 15 milliseconds to say. The analysis process would divide this occurrence into 75 time segments. Since there can not be an exact one to one correspondence between the original 50 segments stored in the template and the newly acquired 75 segments, allowances for time "slippage" must be made. The allowance for slippage is achieved by "Dynamic Time Warping". A more complete mathematical model for DTW can be found in the literature and will not be repeated here [13].

3.1.2 Hidden Markov Modelling

Hidden Markov Modelling is a statistical method which also models the variations in the way a speaker pronounces a word. Each word is divided into states and a probabilistic determination is made on the likelihood of transitions from one state to another. The states in a typical application are generated by filtering and digitizing the input signal, then segmenting it into smaller overlapping blocks which are fit to a linear predictive coding model [13]. Once this has been done a template is created for each word or sound that must be recognized and the incoming signal is analysed and compared to the pre-existing templates. However because the HMM is a statistically based process, the templates must be built from a number of repetitions by the speaker during the learning process [13]. For a given word to be learned

it may require 40 or more repetitions of the word to be spoken in order to produce a template which accurately maps the possible state transitions [16].

Although there are differences between the DTW and HMM methods, good implementations of either of the two processes can result in good speech recognition systems. The trade offs between the two methods come from the decreased learning time require for the DTW method versus the shorter processing time required for the HMM method. DTW algorithms are computationally expensive, since each incoming signal pattern must be compared to those stored in all of the templates [12].

3.2 Warning Signal Characteristics vs Voice/Word Characteristics

The two methods described above have been designed to learn and recognize human speech. This is a much more complicated process than what is required for the recognition of warning sounds. Human speech is not uniform between speakers or even for the same speaker. It is formed by the interaction of the vocal cords, the pharynx, mouth, nose and sinus cavities. The tongue, teeth and lips also interact to produce the sounds we recognize as speech. The same word will sound different depending on its use, the mood of the speaker and even the time of day.

Warning sounds, on the other hand, are mechanically generated and must remain relatively constant in frequency composition and duration over their lifetime in order to perform their intended function. For example, phone rings are recognized throughout Canada and the United States regardless of geographical location because their frequency and timing characteristics are held within narrow limits. The warning sounds generated by other equipment also have characteristics which, in general, do not change significantly, if at all, after installation. Figure 3.1 shows the spectrum for three occurrences of the word "hello", spoken

by the same speaker. Figures 3.2 through 3.4 shows the spectrum for a number of electronically and mechanically generated warning signals.

These graphs show why the learning and recognition processes for warning sounds should be much less complex than those for speech, and why the methods such as DTW and HMM, although well suited to speech recognition, are not the ideal processes for warning sound recognition.

The warning signals shown in Figures 3.2 through 3.4 have characteristic peaks that remain well defined and constant in frequency. They also have burst and interval durations that remain constant in time. From examining these graphs it is possible design a warning signal recognition algorithm. In order to detect a warning signal we must correctly identify the three characteristic features of the signal. For all warning signals, the first feature, the signals characteristic peak frequency, must be detected.

Next we must identify the timing characteristics of the signal. For burst type signals, the burst duration must be determined and if the signal is of repetitive burst type, the interval time between bursts. Should a warning signal be of the continuous type, then we must determine if the signal we are detecting is actually continuous or simply due to noise by ensuring that the signal is detected for a long enough time period.

The third characteristic of a signal is its energy content. Additional information about a warning signal is conveyed by the amount of energy contained in the signal. A louder signal is, in general, considered to be more urgent or important than a softer one. This is a matter of perception, and is highly dependent upon the surrounding environment since "loudness" is usually compared to some arbitrary background noise level. Often just the presence of an increase in sound level can be enough to alert us that something has occurred.

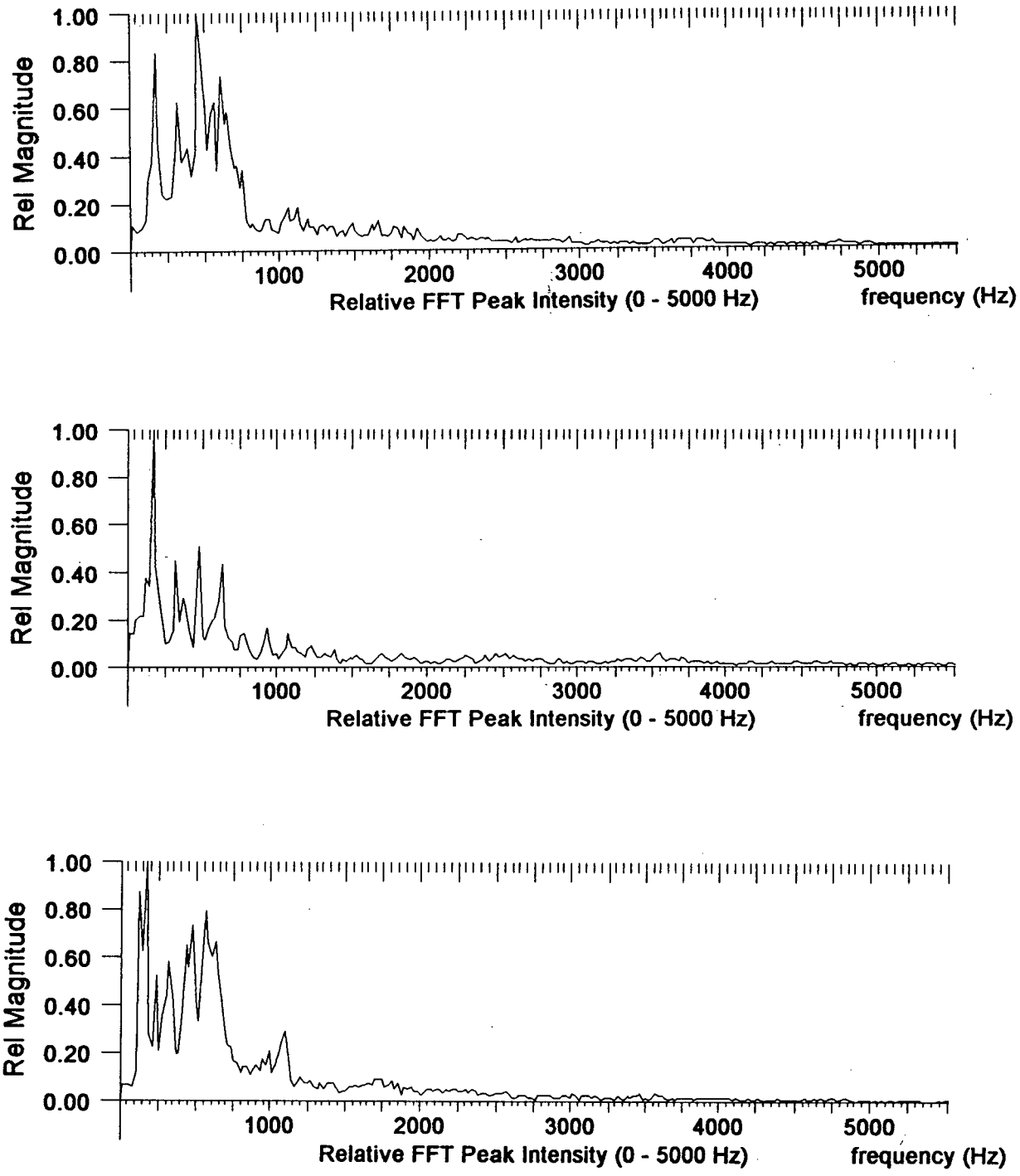


Figure 3.1 - "Hello" Spectral Analysis

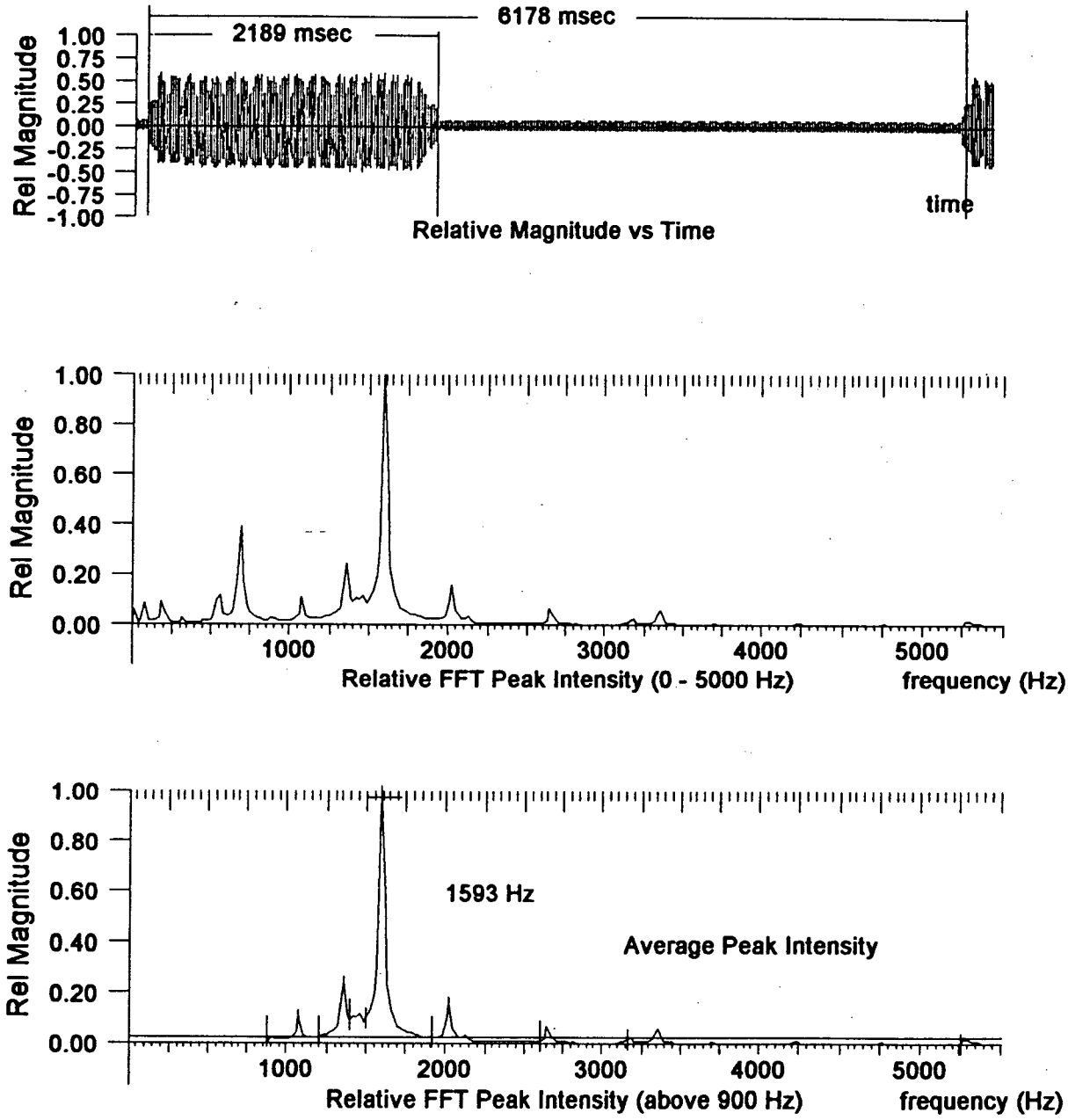


Figure 3.2 - Electronic Ringing Signal Spectral Analysis

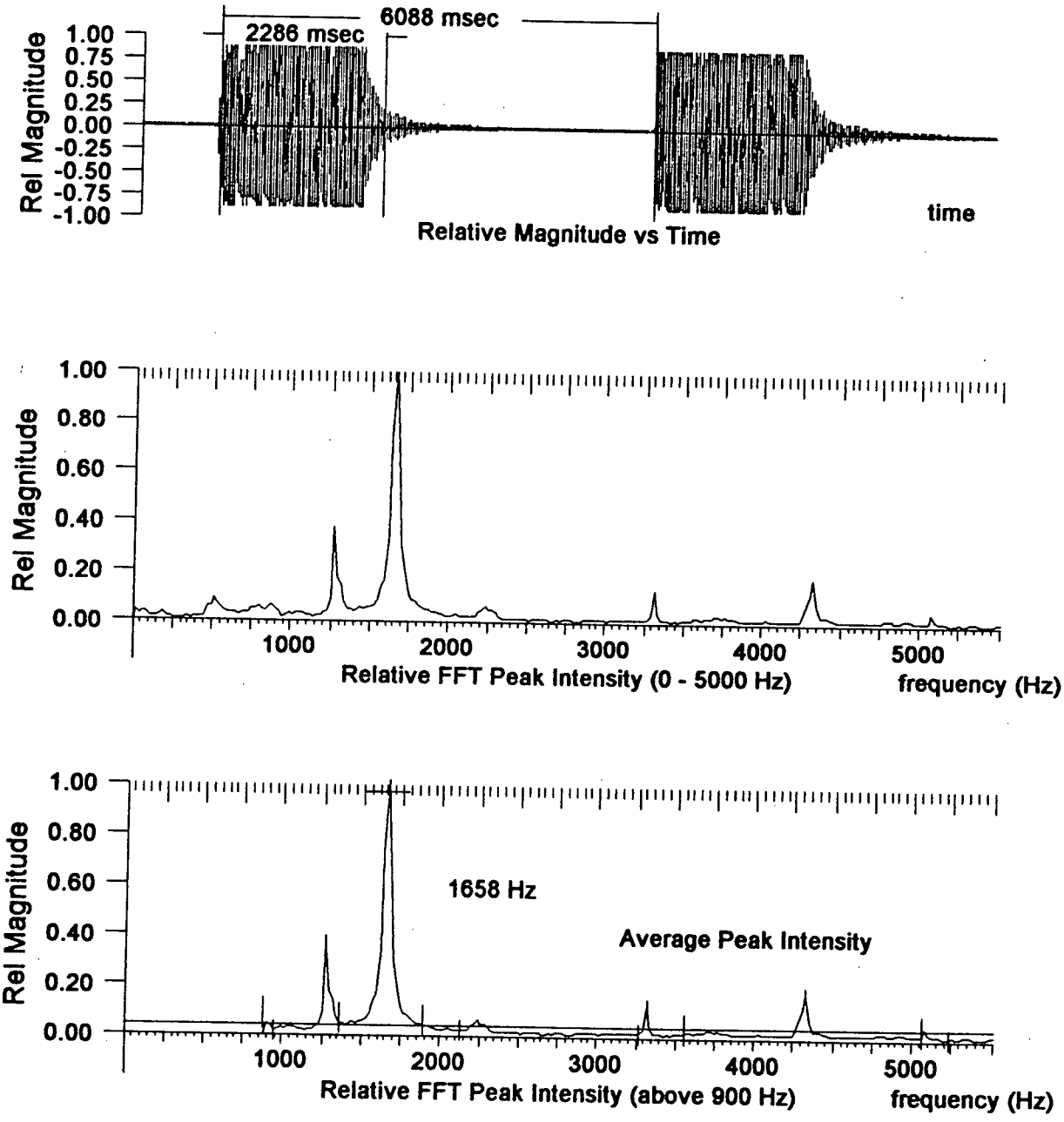


Figure 3.3 - Mechanical Ringing Signal Spectral Analysis

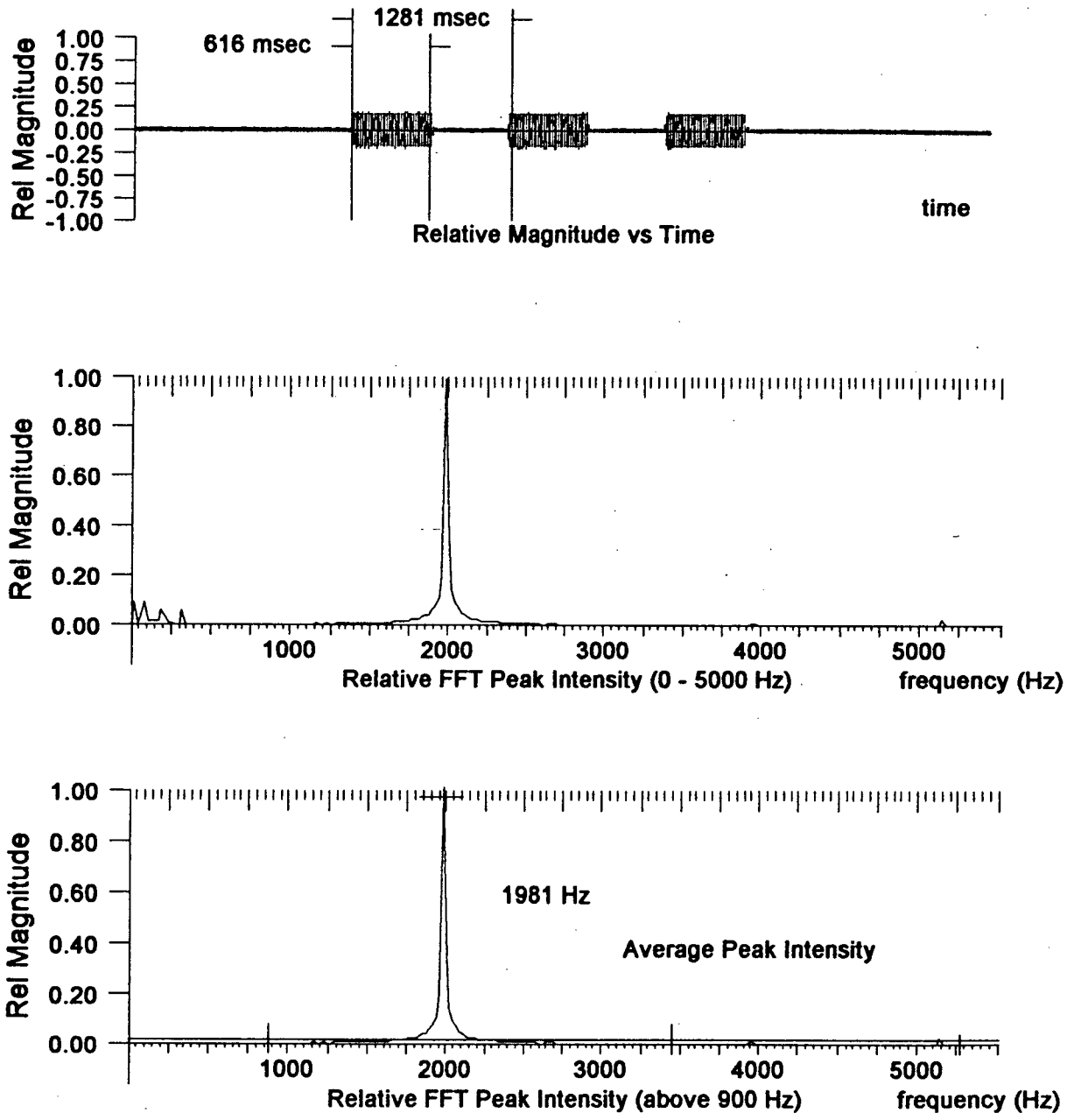


Figure 3.4 - Microwave Alarm Signal Spectral Analysis

This leads us to a 3-part analysis scheme for our recognition algorithm, one part for spectral analysis, a second for timing analysis and a third for energy content analysis. The important question here is: in what order do we perform these analyses? Although frequency , timing and energy characteristics are important, which is most important for recognizing a warning signal?

When a telephone initially begins to ring, one is alerted to that fact by the presence of the ringing sound. Before the ringing burst finishes, it can be determined that a telephone call is incoming. If one is unsure that the telephone actually rang, one can wait for the burst to end or a second burst to repeat for confirmation. It is, however, the presence of the characteristic ringing frequency, where there was none before, that gives the first alert. Similarly, in an office or home, if two closely placed different phones ring simultaneously, the only way to tell them apart is by the different sound of the two ringers.

Even for an unknown warning signal, it is often the presence of a new spectral component in the background noise that gives the warning. When a component of a vehicle breaks down, there is often a new "sound" to the vehicle that is a direct result of the fault. For example, should a bearing wear out, it is often accompanied by a screeching sound. One may not have heard the sound before, but it is immediately obvious that something has happened. It would appear that a significant amount of information required to recognize a warning signal is contained in the spectral and energy characteristics of the signal. Also, although additional information is contained in the timing characteristics of a warning signal, those characteristics would appear to be secondary. For these reasons, for WARNSIS II, we shall approach our algorithmic design with the intention of performing the spectral and energy analyses first.

Chapter 4

4.0 Spectral Analysis Methods

For WARNSIS II, it was decided to implement as much of the analysis as possible, including the spectral analysis, in software. The reasons for this are three-fold. First, by relying on software for as much of the recognition scheme as possible, the overall chip count for the device can be minimized. This allows for lower overall cost for both design and manufacture. By reducing the number of chips or specialty chip sets, both the number of parts and the overall size of the board can be reduced. This in turn will also lower assembly and testing costs. By reducing the parts count the time required to design and layout the circuit is also greatly reduced. Eliminating the use of specialty chips is important. The use of specialty chips would limit the functionality and flexibility of the device because, should improvements to the recognition algorithm be developed in the future, they would not be easy to incorporate into the existing hardware. With software, this is not problem.

Secondly, implementing WARNSIS II in software means that large economies of scale can be applied to the production of the final device. With no specific hardware requirements, any commercially available platform with enough processing power could be used. For example, a PC sound card with production volumes of millions of units per year could be used as the basis for implementing some versions of the WARNSIS II system.

Lastly, although the WARNSIS II project is intended to produce a design which can be implemented as a stand-alone device, the functionality of the WARNSIS II recognition scheme could provide useful features to be incorporated into other devices such as medical monitoring equipment. A software approach makes this "generalization" much easier.

Chapter 4 - Spectral Analysis Methods

4.1 WARNSIS II Spectral Analysis

For WARNSIS II we propose to perform the spectral analysis via a computational method. There are a number of reasons for this approach. First, by using a computational method, such as the Fast Fourier Transform (FFT), the number of frequency ranges and the total frequency range covered can be adjusted by software on an individual signal or implementation basis. This allows for a multilevel spectral analysis, if required. For example, an incoming signal could first be examined for energy content above or below a given value in order to reduce the search path and execution time. Alternatively, the FFT approach would allow for finer or coarser frequency determinations in changing noise environments.

The use of a computational spectral analysis method also allows for some type of noise cancellation. The spectra for the templates obtained during training of the device are likely to be obtained in the "best" conditions available. This could mean no, low or modest noise environments, depending on where the signal was "learned". The spectra of signals detected in the real world are likely to have a large noise content . Noise cancellation on both training and analysis would give the cleanest signal both for the templates and for comparison, and should improve the ability of our scheme to recognize a signal in noisy environments.

There are a number of computational frequency analysis methods in addition to the FFT, mentioned above, which can be used for spectral analysis. Other computational methods such as the Wavelet, Hartley and Cosine transforms, and Finite and Infinite Impulse Response filters (FIR and IIR) are available. These will be compared in the following sections to determine which method best fits the WARNSIS II requirements.

4.1.1 Fast Fourier Transform [17,18,19,20,21]

The FFT, first popularized by J.W. Cooley and J.W. Tukey in 1965 [18,21], is currently one of the most common and successful methods for performing spectral analysis on digitized data.

Suppose we a sample a continuous function of time h(t) so that we have N consecutive values, where N is even, with a sampling interval of Δ.

$$h_k \equiv h(t_k) , \quad t_k \equiv k\Delta , \quad k = 0,1,2, \ldots , N-1 \qquad (4.1)$$

If we are attempting to find the Fourier transform for this function, then having N input values we can produce no more than N independent output values. This means that we can

$$f_n \equiv \frac{n}{N\Delta}, \qquad n = -\frac{N}{2}, \ldots , \frac{N}{2} \qquad -\frac{N}{2} \equiv \frac{N}{2} \qquad (4.2)$$

obtain estimates only for those frequencies at discrete values or *bins*. We approximate the Fourier integral by the sum

$$_n) = \int_{-\infty}^{+\infty} h(t)\, e^{2\pi i f_n t}\, dt \approx \sum_{k=0}^{N-1} h_k e^{2\pi i f_n t_k} \Delta = \Delta \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N} \qquad ($$

The final summation in equation (4.3) above is known as the Discrete Fourier Transform (DFT). The DFT requires N^2 complex multiplications and is an $O(N^2)$ process. Using the

FFT algorithm, the Discrete Fourier Transform can be calculated on the order of $O(N\log_2 N)$, which is an enormous difference. The FFT is derived as:

$$\sum_{j=0} e^{2\pi ijk/N} f(j)$$

$$\sum_{j=0}^{(N/2)-1} e^{2\pi ik(2j)/N} f(2j) + \sum_{j=0}^{(N/2)-1} e^{2\pi ik(2j+1)/N} f(2j+1)$$

$$\sum_{j=0}^{(N/2)-1} e^{2\pi ikj/(N/2)} f(2j) + W^k \sum_{j=0}^{(N/2)-1} e^{2\pi ikj/(N/2)} f(2j+1)$$

$$F^e + W^k F^o$$

where $W^k = e^{2\pi i/N}$, sometimes referred to as the "*twiddle factor*" ; F_k denotes the k component of the Fourier transform of length N/2 formed from the even components of the original $f(j)$ and F_k^o is the corresponding transform of length N/2 formed from the odd components. The fact that this formula can be used recursively means that, for transforms for which N is an integral power of 2, it can be repeatedly broken down until we have only to calculate transforms of length 1. A Fourier transform of length 1 is simply the identity operation that copies it input value to its output slot or *bin*. This means that every length 1 transform in the series corresponds to one of the input values. The correct input value can be determined by bit reversing the index j for the original function. If you take the original data sequence and rearrange it so that the values are in order not of *j*, but in order of the numbers obtained by reversing the bits of *j* written as a binary number, you have the correct relation between the inputs and the length one transforms. For example, if *j=3* for a length 8 transform then the correct output bin for the length one transform would be given by:

$$j = 4_{10} = 100_2 \quad \text{yielding} \qquad \text{bin \#} = 001_2 = 1_{10} \qquad (4.5)$$

From this it is a simple matter of calculating the combined transforms of size 2, 4, ..., N in recursive stages. Figure 4.1 shows a length 8 transform being bit reversed and then recombined.

4.1.1.1 FFT and Spectral Power Estimation [19]

Since we are trying to determine if a signal being analysed by WARNSIS II has a spectral component at a given frequency, we are interested in the energy content of a signal at that frequency. This means that rather than using the Fourier Transform itself, as we would if we were required to perform a cross-correlation with a second signal, we require only the signal's power spectrum. This can be done using the FFT. Rather than matching the power at a particular frequency f_k that would be obtained from calculating the continuous Fourier transform, the power given by the FFT is an "average" or expected value for the frequency *bin* centred at f_k extending over a narrow window . Historically this has been called a *periodogram*. For an N-point sample of a function c(t) using the FFT

$$C_k = \sum_{j=0}^{N-1} c_j e^{2\pi i j k/N} \qquad k = 0, \ldots, N-1 \tag{4.6}$$

the periodogram estimate of the power spectrum is defined as

$$\begin{aligned} P(0) &= P(f_0) = \frac{1}{N^2}|C_0|^2 \\ P(f_k) &= \frac{1}{N^2}\left[\,|C_k|^2 + |C_{N-k}|^2\,\right] \quad k = 1, 2, \ldots, (\frac{N}{2} - 1) \\ P(f_c) &= P(f_{N/2}) = \frac{1}{N^2}|C_{N/2}|^2 \end{aligned} \tag{4.7}$$

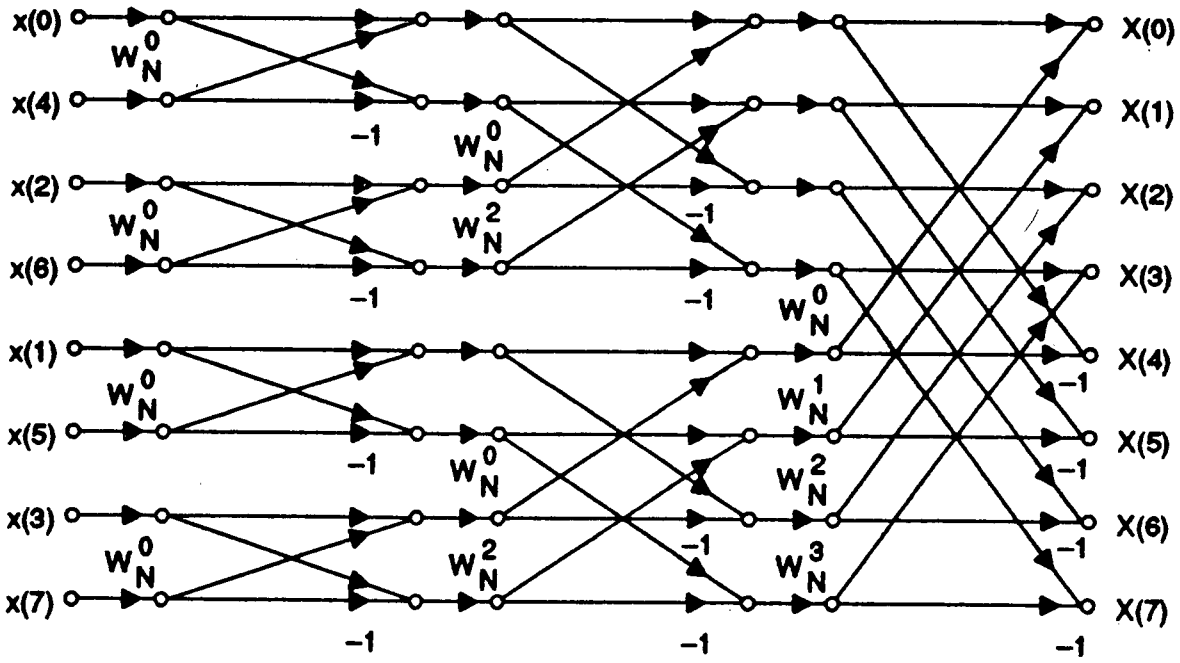


Figure 4.1 - FFT Bit Reversal Recombination

where f_k is defined only for the zero and positive frequencies

$$_k \equiv \frac{k}{N\Delta} = 2f_c \frac{k}{N} \quad k = 0, 1, \ldots, \frac{N}{2} \qquad (4.$$

For the periodogram defined above, the narrow window over which the average is taken, as a function of s the frequency offset in bins, is described by

$$W(s) = \frac{1}{N^2}\left[\frac{\sin(\pi s)}{\sin(\pi s/N)}\right]^2 \qquad (4.9)$$

This window has side lobes that contribute significant amount of "leakage" a number of bins away.

4.1.1.2 Data Windowing

Performing a Fourier transform on a given function of time assumes an infinite sequence of input values for the integral. The FFT, which is used for discrete data, is always performed on a sample size somewhat less than this. By performing an FFT on a finite sample size we are in effect convoluting an "infinite" sequence with a square window function which is 0 except during the finite sampling time. The overall effect of the finite sample is to cause "leakage" in the power spectrum from the true frequency to other frequencies a number of *bins* away as described above. The "leakage" can be substantial even at frequencies a large number of bins away from the frequency of interest because the square window function, in effect, turns on and off instantaneously.

To overcome this leakage problem it is possible to modify the sampling window function by multiplying the input signal data c_j , j - 0, ... N-1 by a windowing function w. There have been a number of alternative window functions proposed [19]:

the Parzen window,

$$w_j = 1 - \left| \frac{j - \frac{1}{2}(N - 1)}{\frac{1}{2}(N + 1)} \right| \qquad (4.10)$$

the Welch window ,

$$w_j = 1 - \left| \frac{j - \frac{1}{2}(N - 1)}{\frac{1}{2}(N + 1)} \right| \qquad (4.11)$$

and the Hanning window,

$$w_j = \frac{1}{2} \left| 1 - \cos\left(\frac{2\Pi j}{N-1}\right) \right| \qquad (4.12)$$

Each of these window functions is shown in Figure 4.2 [19]. The effect that each of the windowing functions have on the "leakage" between bins is different. These effects are usually given "figures of merit" that can be used to describe the narrowness of the peaks. The tradeoff between different window functions is between the narrowness of the peak and the fall off of the tails. In general it is recommended, that if a windowing function is required, the

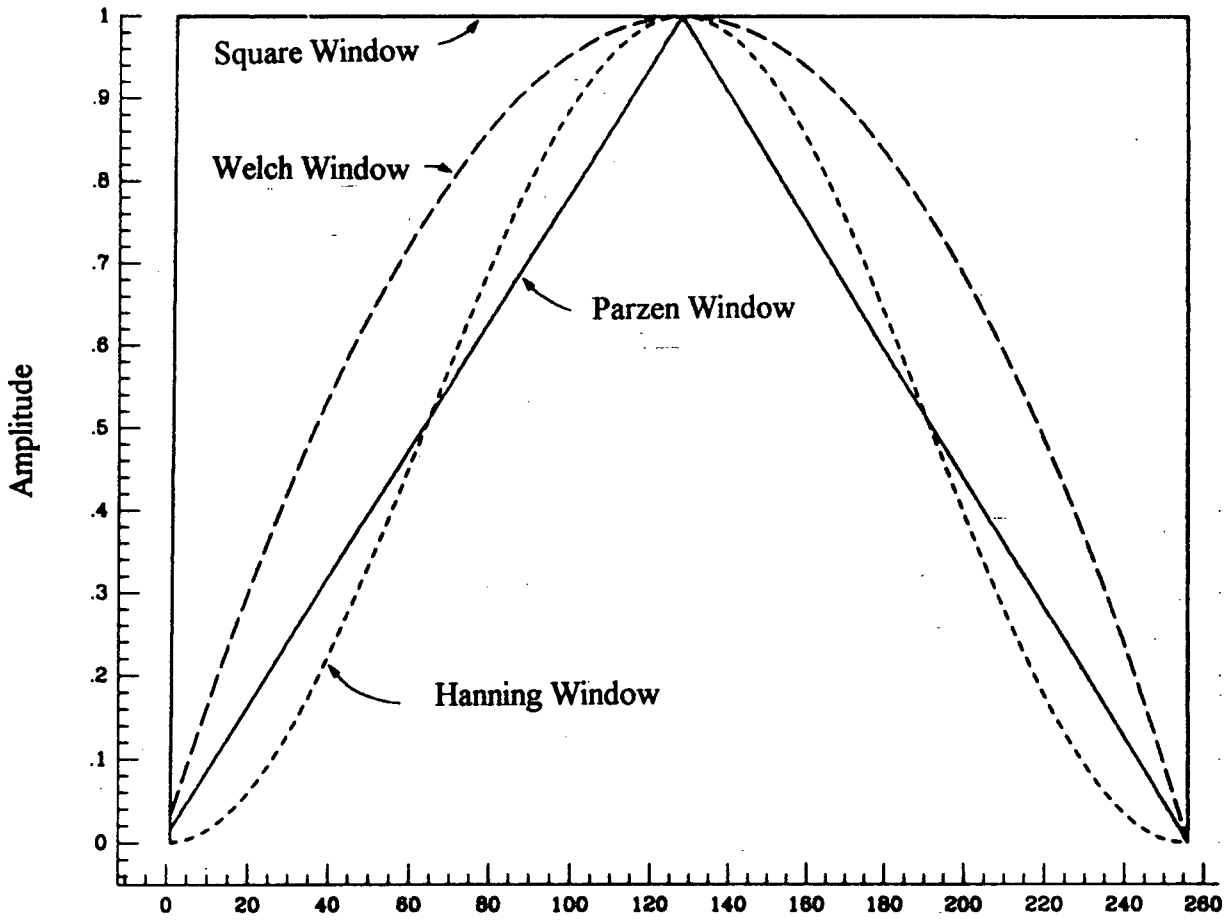


Figure 4.2 - Parzen, Welch & Parzen Window Functions

Parzen or Welch windows be used. The larger computational effort required for the cosine function in the Hanning window gives minimal additional value [19]. Figure 4.3 shows the effects of the three windowing functions in comparison to the square sampling window [19].

4.1.1.3 Maximum Entropy Method [19,32]

As stated earlier, when performing a spectral analysis using the FFT there is a relationship between the number of samples that must be used to produce a given frequency resolution for a given sampling rate. The finer the frequency resolution, the larger the number of samples that must be considered. This can lead to prohibitively large sample sizes which can consume large amounts of memory while processing.

The Maximum Entropy Method (MEM) [19, 32] for power spectrum estimation allows one to perform a parametric (model based) spectral analysis which will provide a better frequency resolution for a smaller sample size. The MEM is an auto-regressive method which characterizes a known signal in terms of a finite number of poles that best represents its spectrum in the complex plane.

The MEM can produce a better spectral resolution for a smaller number of samples than the standard FFT methods. The MEM does, however, have a few disadvantages. Firstly, it exhibits line splitting at high signal to noise ratios. Secondly, at higher orders the MEM can introduce spurious peaks. Thirdly, for sinusoidal signals in noise, which would encompass a large number of the warning signals of interest to a WARNSIS device, there is a frequency shift in the signals spectrum that is dependant on the initial phase of the signal which we are trying to recognize. For these reasons and because the MEM requires extra computational effort to calculate the parametric coefficients, this method was not used for WARNSIS II.

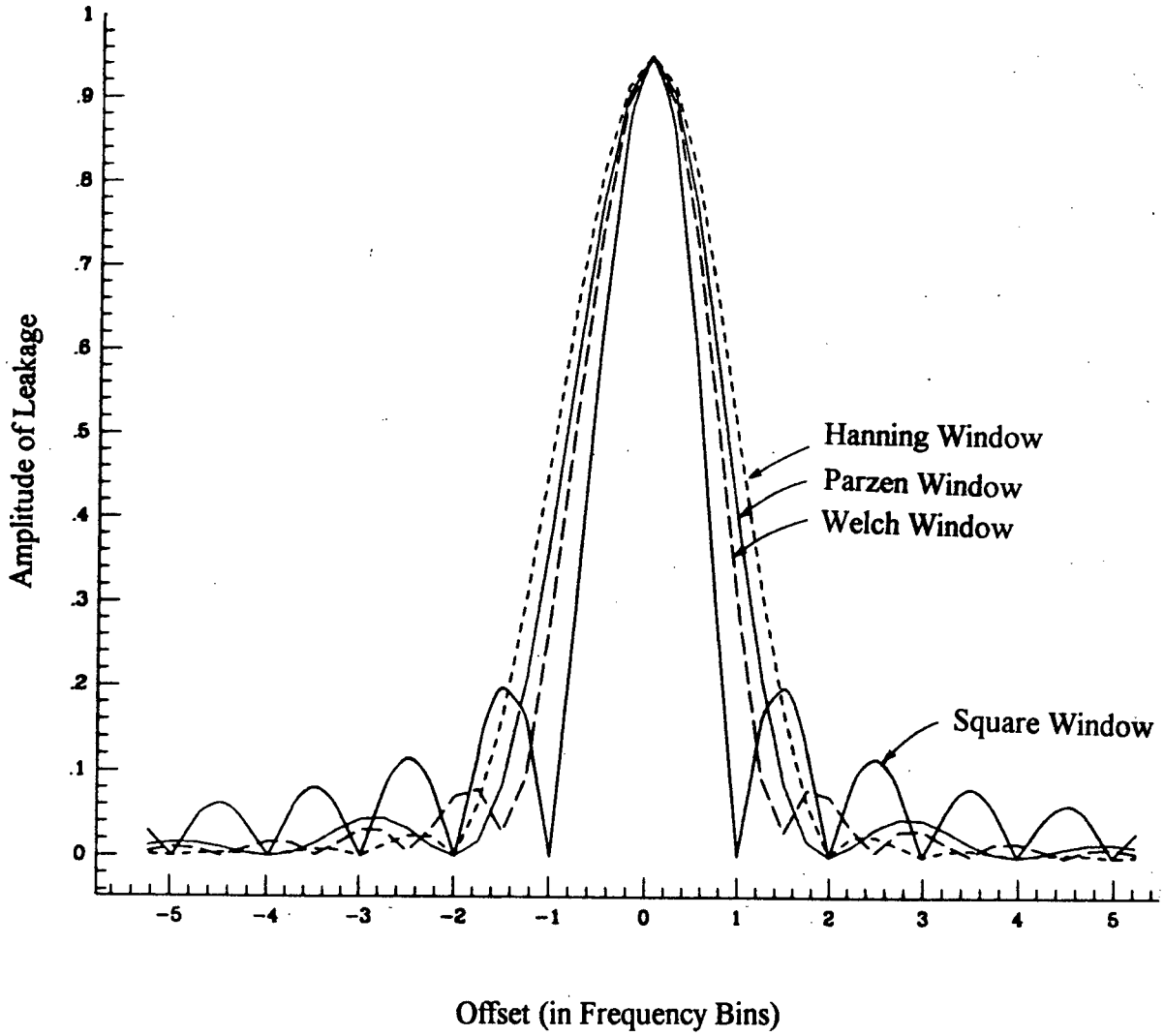


Figure 4.3 - Leakage for Parzen, Welch & Hanning Window Functions

4.1.2 Discrete Hartley Transform [17,22,23]

The Discrete Hartley Transform (DHT) is applicable to real valued signals and is closely related to the FFT , in fact the DHT and the FFT can be derived from each other . The DHT is defined as:

$$H(K) = \sum_{n=0}^{N-1} x(n)\cos(2\pi k\, n / N) + x(n)\sin(2\pi k\, n / N) \quad k = 0,\ldots, N-1 \quad (4.13)$$

$$x(n) = 1/N \sum_{k=0}^{N-1} H(k)\cos(2\pi k\, n / N) + H(k)\sin(2\pi k\, n / N) \quad k = 0,\ldots, N-1 \quad (4.14)$$

where x(n) is the real valued sequence. There have been a number of claims that the Fast Hartley Transform (FHT) implementation of the DHT is more efficient than the real-valued FFT for computing the power spectrum for real-valued sequences with $N = 2^P$. The two most recent comparisons of the two methods have concluded that the FFT is in fact a more computationally efficient algorithm [22,23]. The time required for computing the FFT vs the time required for computing the DHT give a ratio of processing times of 1:1.04 for a size 512 transform [24]. For the TMS320C30 processor used in WARNSIS II, the ratio of processing time between implementations of the FFT and DHT on the same signal processor is approximately 1:1.98 in favour of the FFT which is a significant difference [17].

4.1.3 Discrete Cosine Transform [17,19,24]

The Discreet Cosine Transforms (DCT) also arises from the FFT. The DCT is given by:

$$F(K) = \sum_{n=0}^{N-1} x(n) \cos(\pi k\, n / N) \qquad k = 0,..., N-1 \qquad (4.15)$$

where x(n) is the real valued sequence. Although the DCT appears to be simply the real part of the Fourier Transform, it differs by a factor of 2. The DCT and Discrete Sine Transforms (DST) are more useful when applied to boundary value problems than for generating the power spectrum information that we require. Although new, more efficient, direct computation algorithms have been proposed for the DCT and DST [24], the FFT is still superior for our purposes. The ratio of processing time for implementations of both the FFT and DCT on the TMS320C30 signal processor is approximately 1:1.91 in favour of the FFT for a size 512 transform [17].

4.1.4 Finite and Infinite Impulse Response Filters [17,19,20]

Rather than using the FFT or other similar transform, it is possible to duplicate the process used in WARNSIS I where the frequency spectrum was divided into 8 ranges by the filter bank by the use of linear digital filters. If we assume that the signal x(n) is applied to a bank of complex filters with impulse response $h(n)\exp(j\omega_k n)$ where ω_k is the centre frequency of any filter, and h(n) is the response of an ideal lowpass filter with bandwidth ω_h. If the real part of the filter output is squared, doubled and bandlimited to $2\omega_h$, then the resulting analysis is the same as the square of the magnitude of the Short-Time Fourier Transform of x(n) for $2\omega_h < |\omega_k| < \pi - 2\omega_h$ [20].

The generic linear filter is formed by taking the sequence of inputs x_n and producing a series of outputs y_n such that

$$y_n = \sum_{k=0}^{M} c_k x_{n-k} + \sum_{j=1}^{N} d_k y_{n-j} \qquad (4.16)$$

where c_k and d_j are fixed coefficients which define the filter response. If N=0 so that the second sum does not contribute to the filter response, then the filter is said to be *non-recursive* or a *finite impulse response* filter (FIR). If N<>0 then the filter produces a response from not only the current input values but from its own previous output as well. In this case the filter is said to be *recursive* or an *infinite impulse response* filter (IIR).

The main drawback of using these methods is in the calculation of the coefficients. There are a number of methods for calculating the coefficients c_k and d_j depending on whether one is implementing an FIR or IIR. The *Remez Exchange Algorithm* produces the best Chebyshev approximation for a fixed number of filter coefficients for an FIR and one useful technique for IIR's is the *bilinear transformation method* [19]. The drawback is that all of the methods for calculating the coefficients require large amounts of processing time. This would then require WARNSIS II to use higher speed and higher cost processors to implement the spectral analysis component of the WARNSIS II algorithm. A second drawback of the filter approach is that even though new IIR algorithms are available for tracking sinusoidal signals in noisy environments, the algorithms all exhibit "rather erratic" convergence behaviour [25].

4.1.5 Wavelet Transform [26,27,28]

The Wavelet Transform (WT) is an alternative time-frequency representation that can

be used for spectral analysis. The time-frequency version of the WT is defined as

$$WT_x^{(\gamma)}(t,f) = \int_{t'} x(t') \sqrt{|f/f_0|}\, \gamma^* \left| \frac{f}{f_0}(t'-t) \right| dt' \qquad (4.17)$$

where $\gamma(t)$ is the "analysing wavelet" and is a real or complex *bandpass* function centred around time t=0 and f_0 is the centre frequency of $\gamma(t)$ [26,28]. In principle the wavelet transform and the Fourier transform are very similar in their approach to time-frequency representation, as both can be represented as inner products. In fact the WT is just a special case of the generalized STFT [27]. There are two essential differences between the wavelet and Fourier transforms. The first is the choice of a linear time-shift/frequency-shift operator for the Fourier transform versus a time-scaling/time-shift operator for the wavelet transform. The second is that the equivalent $\gamma(t)$ function for the Fourier transform is a lowpass function instead of a bandpass function for the wavelet transform.

Both transforms suffer from similar time-frequency resolution limits. Thus, neither transform can provide arbitrarily good time resolution and frequency resolution simultaneously. For example using the Fourier transform with a given sampling rate, if one wishes to increase the frequency resolution of the power spectrum, one must increase the number of samples taken, thus producing a power spectrum covering a longer time period. However, the Fourier and Wavelet Transforms are different regarding their time-frequency resolution in ways which are important for the WARNSIS II algorithm. In particular, for the Fourier Transform the time-frequency resolution is the same for all frequencies. For the Wavelet Transform, however, the time-frequency resolution is not constant, higher frequencies are analysed with better time resolution, but with poorer frequency resolution [28]. This is unacceptable for the WARNSIS II algorithm since it is essentially the higher frequencies that are of interest. The

time-frequency resolution should be constant over the entire frequency range of interest since we have no way of predicting *a priori* which frequencies will be relevant when the device is actually in use.

4.2 Evaluation of Computational Methods for Spectral Analysis

The FFT approach to frequency analysis was chosen over the WT for the reasons explained above. The FFT is also preferred to the Hartley, and Cosine Transforms and the FIR and IIR filter methods because of the time comparison factors as discussed in sections 5.1.2, 5.1.3 and 5.1.4, as well as other reasons. Firstly, the FFT is one of the best understood computational methods and optimized code for performing the FFT is available for most computer languages and systems available on the market today. Also, because of its popularity, many of the "off-the-shelf" signal processing chips are architecturally designed to exploit fast addressing schemes which provide for a further increase in FFT processing speed.

Additionally, further speed increases can be gained by exploiting specialized implementations of the FFT algorithm which are known as RADIX-2 or RADIX-4 versions. When the number of input values for the FFT is n^2 or n^4 then the FFT processing speed can be increased by reducing the number of stages used. Also for the WARNSIS II algorithm we are digitizing real signals, hence the input to the FFT routines will be real values with no complex parts. This allows us to take advantage of the ability to double the FFT processing speed by substituting real values for complex values in the input array.

The FIR and IIR methods were not considered as the number of filter banks required to implement a fine enough frequency resolution would be computationally prohibitive.

Chapter 5

5.0 WARNSIS II Design

5.1 Overview

The WARNSIS II warning sound recognition system is a set of software algorithms which can be implemented on any hardware platform with sufficient processing power to perform the required real-time FFT analysis and template matching algorithms. The system consists of a number of functional modules which allow the WARNSIS II system to operate in two modes: learning mode and signal recognition mode. In the learning mode, a template for a warning signal is "learned" by analyzing the signal for spectral and temporal information. The frequency(s) at which the maximum peak(s) in the signal's spectral energy occur are determined and these values are saved in a template. Timing information, specifically, the duration of a warning signal burst and the interval time between warning signal bursts (for repetitive signals such as telephone rings) are determined and also saved in the template.

In the recognition mode, the spectrum of the incoming real time signal is continuously analysed. The analysis determines if any of the incoming signal's spectral components match those stored in one of the "learned" templates. If a spectral match is found, timing information is then applied to rule out transients which could generate false alarms.

5.2 Learning Mode

As stated earlier, the drawbacks of the devices commercially available today are their lack of flexibility causing difficulties when moving from one location to another and their lack of the ability to accommodate new types of warning signals. To overcome these deficiencies a WARNSIS II device must include the capability for the automatic "learning" of a new signal.

When the learning mode is entered, a signal learning algorithm begins. The input signal is continuously monitored until an increase in the Short Time Average Absolute Amplitude (STAAA) is found. The STAAA is calculated by averaging the absolute value of the input amplitude over a period of 512 samples (63.9 ms at 8,012 Hz sampling rate). This is sufficient to eliminate short noise bursts which would interfere with the second stage of the signal detection algorithm. When the learning mode is first entered a sample of the background sound level is taken and its STAAA is calculated. This level then becomes the value against which all later STAAA's are compared in order to determine if an increase in signal amplitude due to a warning signal has occurred. If a future STAAA is greater than 1.5 times the background level value then a valid amplitude increase has been detected. To actually detect the beginning of a valid signal, the signal acquisition algorithm requires that two consecutive STAAA's 1.5 times greater than the background level be detected. If a valid STAAA is detected, but the next STAAA fails to exceed 1.5 times the background value, then the first STAAA is ignored. This differs from the original WARNSIS I algorithm in that there is no Dynamic Amplitude Threshold (DAT) and that two consecutive valid STAAA's are required to initiate the learning process. Although this algorithm would be affected by an overall increase in ambient noise levels, causing it to falsely detect the start of a warning signal, it should be possible in most cases to "learn" a signal in a low noise environment. For example, it is possible to lower the ambient noise level in a home or office for a short period of time by temporarily eliminating the noise sources. Figure 5.1 shows the learning mode process described above.

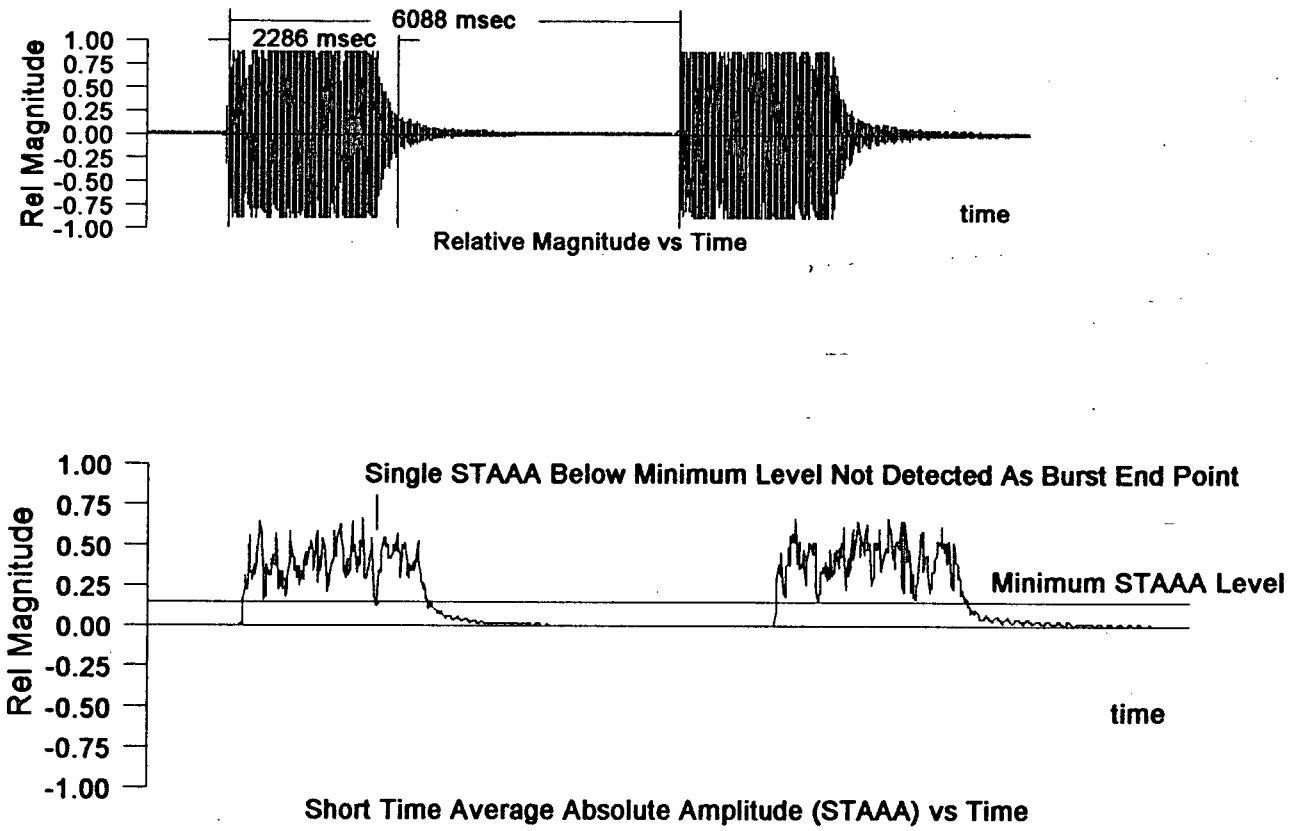


Figure 5.1 - Short Time Average Absolute Amplitude (STAAA) for Burst Determination

Once two consecutive STAAA's have been detected, the signal acquisition algorithm sets the start of valid signal flag and begins the spectral analysis. The spectral analysis at this stage of the learning process is performed by continuously calculating the FFT on the incoming signal in 512 sample intervals. As each set of 512 samples are collected the FFT is processed and the magnitude of the FFT for each *bin* in the power spectrum is calculated and added to the total sum of all magnitudes calculated for each *bin* from all of the previous 512 sample interval since the start of the warning signals acquisition.

This process continues until the warning signal is "lost". The signal is lost if two consecutive STAAA's *fail* to exceed 1.5 times the background STAAA value. If a single STAAA fails to exceed the background STAAA limit but the STAAA from the next 512 sample interval does exceed it, then the signal is not lost. Once the warning signal has been lost, the spectral analysis is ended, the FFT magnitude sum is preserved and the warning signal burst duration is calculated as the time from the detection of the *first* of the two initial STAAA's to the time of the *first* of the two consecutive STAAA failures. Monitoring of the input signal continues until one of two events happens. If two new consecutive STAAA's above the background value are detected before the learning mode "times out", then a valid signal repetition is deemed to have been detected and the signal interval time is calculated as the time from the detection of the *first* of the two initial STAAA's at the beginning of the initial burst to the time of the *first* of the two consecutive STAAA at the beginning of the repetition. If a repetition is detected then the learning mode signal acquisition phase ends. Alternately, if no repetitive burst is noted before the learning mode times out (after approximately 30 seconds) the signal acquisition phase is ended anyway. If the learning mode times out while the signal is still being acquired then a continuous signal, such as that from a smoke detector, is deemed to have been detected.

Once the signal acquisition phase ends and the burst duration and interval have been determined, the FFT magnitude sum is examined to determine the main spectral peak(s). The centre frequency for the *bin* which has the greatest magnitude sum among those *bins* for frequencies above 900 Hz is deemed to be the dominant spectral peak for the acquired signal. The peak is detected by an algorithm similar to the algorithm used for detecting the beginning and end of the signal during the signal acquisition phase. The FFT magnitudes are examined and the start of a peak is determined by two consecutive increases in FFT magnitude when compared with the previous adjacent FFT *bin* magnitude. There is no preset value for the peak increases to exceed as it is only the increase that is relevant, not the absolute value of the increase. As peak detection proceeds, the FFT magnitude values are compared for each bin within a peak. The maximum magnitude found for each peak during this examination process is saved as the peak maximum. The bin number corresponding to the maximum magnitude found is taken as the centre frequency of the peak. Peak detection continues until the peak maximum is detected, again this is determined by two consecutive decreases in FFT magnitude for adjacent *bins*. The "down slope" of the peak is monitored until the FFT magnitude exhibits either two more consecutive increases, indicating the start of another peak, or two consecutive equal magnitudes, indicating the "bottom" of the peak. The width, centre frequency and maximum magnitude are thus determined for each peak in the FFT magnitude spectral sum.

Once the entire spectrum has been scanned and all peaks have been identified, the characteristic peak for the warning signal is determined to be the peak with the greatest peak magnitude. The maximum peak magnitude is used, rather than the peak "area", as warning signals tend to have narrow peaks and noise tends to be broad band covering a larger number of *bins*. A noise peak may contain more total energy than a warning signal peak, but that energy is spread over a greater frequency range for the noise peak than for the signal peak.

The warning signal is thus characterised by three attributes:

i) a characteristic peak frequency (bin number),

ii) burst duration time

iii) burst interval time (if required).

The peak magnitude and width are not currently used for signal recognition and are not stored in the learned warning signal's template.

Figure 5.2 shows the flowchart for the generalized WARNSIS II learning algorithm.

5.3 Recognition Mode

The recognition mode in WARNSIS II operates much in the same manner as learning mode, except that, rather than simply using amplitude increases and decreases for identifying the beginning and ending of signal bursts, it is the presence or absence of a signal's characteristic peak frequency that determines the beginning and ending of a burst.

While operating in recognition mode, the input is continuously monitored and the FFT of the incoming signal is continuously calculated in the same 512 sample size intervals as for the learning mode. It is important that the same sample size and sampling rate be used for both the learning mode and the recognition mode as changing the sample size or sampling rate changes both the size and centre frequency of the *bins*. Once the FFT has been calculated for a 512 sample interval, rather than summing the FFT's output for an extended period as in the

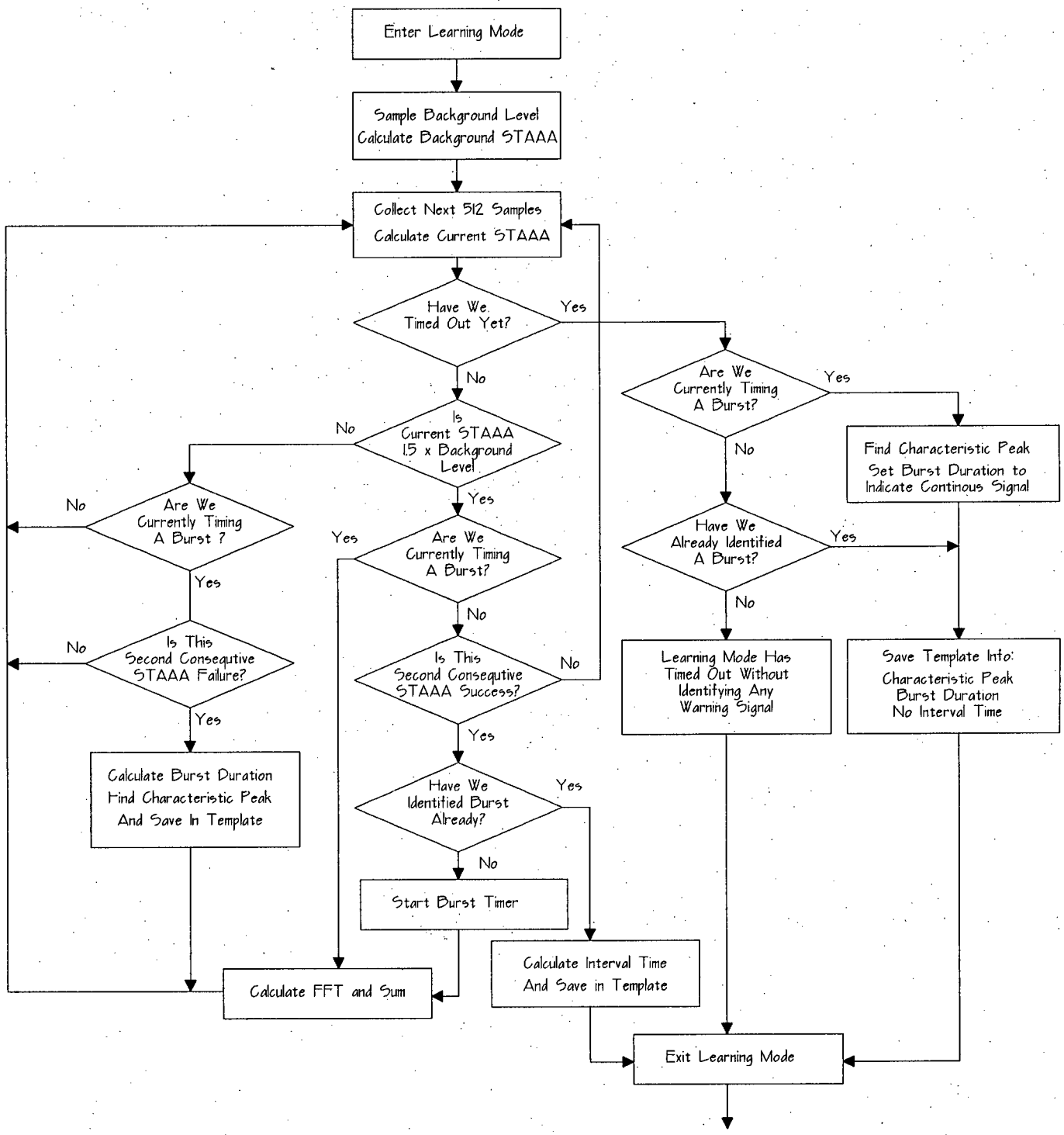


Figure 5.2 - Flowchart for WARNSIS II Learning Mode Algorithm

learning mode, the spectral analysis is performed on the individual 512 sample intervals. As each 512 sample segment is processed, all peaks present in that time interval's spectrum are identified using exactly the same peak detection algorithm as was used in the learning mode.

Each peak above 900 Hz present in the current segment's spectrum that is greater in magnitude than a minimum limit and that is also greater in magnitude than 25% of the magnitude of the largest peak is compared to the characteristic peak frequencies stored in the templates for the various signals that have been learned. The input signal's peak magnitudes are compared to both a preset minimum peak energy value and the 25% limit in order to eliminate false peak detection. Since the energy content in the background signal at frequencies above 900 Hz is generally due to transients and higher harmonics, the magnitude of these peaks are generally lower than those for true warning signals. By only looking at the significant peaks in the spectrum we can eliminate the peaks due to transients. Transient signals are eliminated by this method because signals that are of short time duration with respect to the sampling period, and are of limited power do not contribute as much to the power spectrum as do longer duration, higher intensity signals.

The WARNSIS II algorithm is only interested in the spectrum above 900 Hz and takes no account of any spectral components below 900 Hz. This means that if no warning signal is present, the spectrum of the incoming signal above 900 Hz will be filled with peaks generated by the background noise. Since there will be no large magnitude peak due to a warning signal, the largest peak above 900 Hz will belong to the background noise spectrum and the 25% limit will be set relative to the background noise. This means that noise peaks would then be considered as possible candidates for recognition.

The 900Hz lower frequency limit and 25% magnitude limit were chosen by examining the frequency spectrum from a number of digitized background samples. It was found that in almost all cases the magnitude of peaks in the frequency spectrum decreased significantly at frequencies greater than 900 Hz and that, in general, noise peaks above 900 Hz had magnitudes less than 25% of the magnitude of the signals that we were examining. These are not fixed limits. Since these values as well as all other limits for the WARNSIS II algorithms are set in the software, different implementations of WARNSIS II could use different limits.

To overcome this problem WARNSIS II uses a 2-part level analysis. In addition to ignoring peaks which are less than 25% of the maximum peak detected, WARNSIS II, as stated earlier, also ignores any peak which is lower in magnitude than a preset minimum value. By only "acknowledging" peaks with at least a minimum energy content we can eliminate those which do not contain energy substantially above the background level. This is more effective than using a total signal amplitude threshold approach since in a typical noisy environment that the device may be used, most of the background noise signal's total energy is generated by voices, radio or television broadcasts and will be found below 900 Hz. This 2-part analysis allows a warning signal to be successfully detected by its characteristic frequency peak even in a very noisy environment.

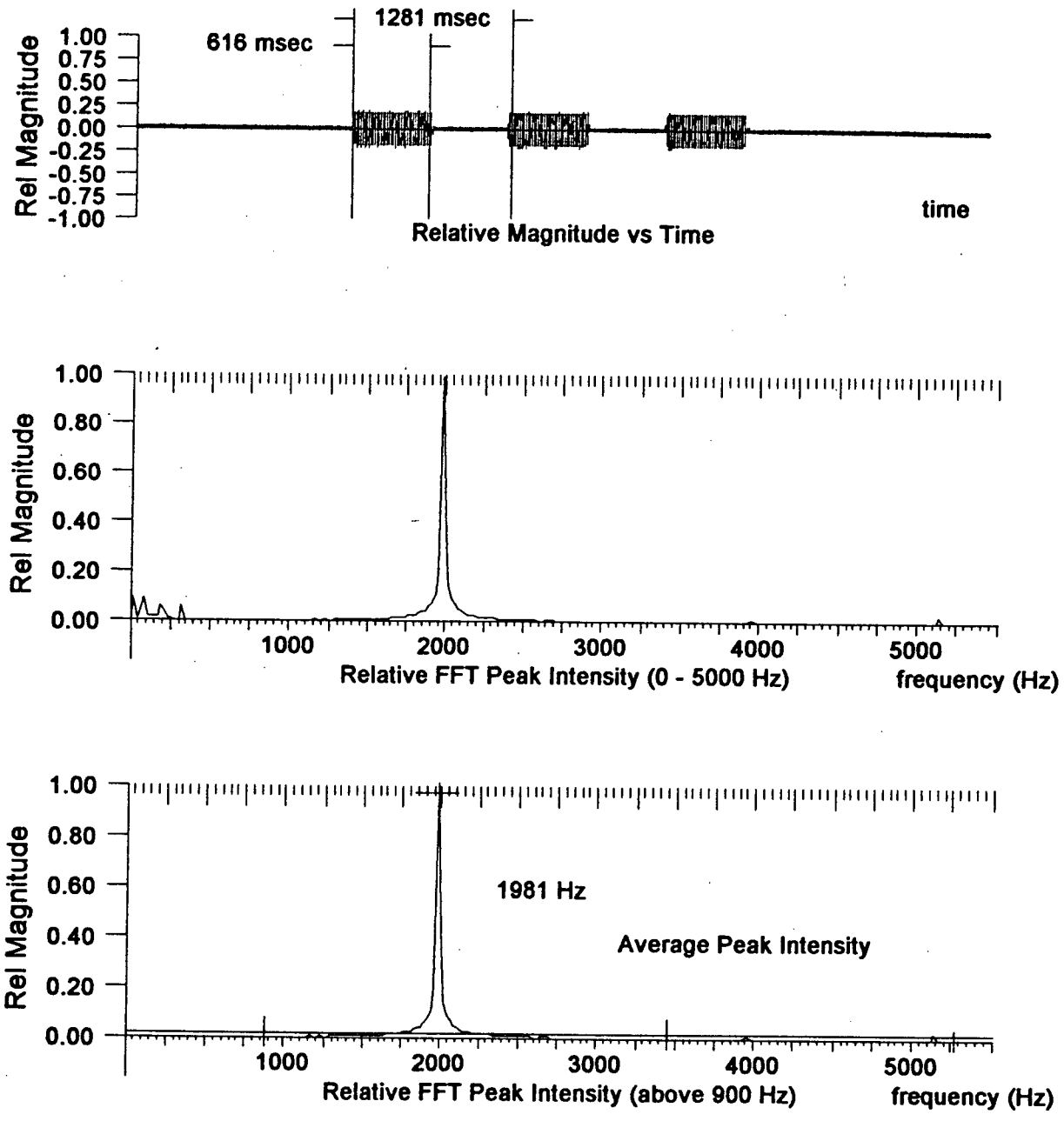
There is one drawback to this 2-part analysis method. If two warning signals occur at the same time, the peak magnitudes will be normalized with respect to the stronger signal. If one of those signals is significantly weaker than the other, so that the weaker signal's peak does not exceed the 25% threshold, the detection of the weaker signal would be suppressed. This means that a signal with a fixed magnitude may not be detected in the presence of a significantly stronger warning signal, even though it would be successfully detected if it

occurred by itself. However, this is the case for the human ear as well, a ringing telephone at a given volume will be detected above the general office or home background noise level, but not above the ring of a fire alarm bell.

Figure 5.3 (a) and (b) show the power spectrum for typical warning signals (in this case a microwave oven alarm (a) and a mechanical telephone ringer (b)). Figure 5.3 (c) shows the power spectrum for typical background speech and music. For each figure, the top graph shows the digitized input signal's magnitude vs time. The second graph in each figure shows the relative magnitude between peaks for the complete power spectrum for the input signal from 0 to 5000 Hz. The third graph in each figure shows the relative magnitude for those peaks in the power spectrum above 900 Hz.

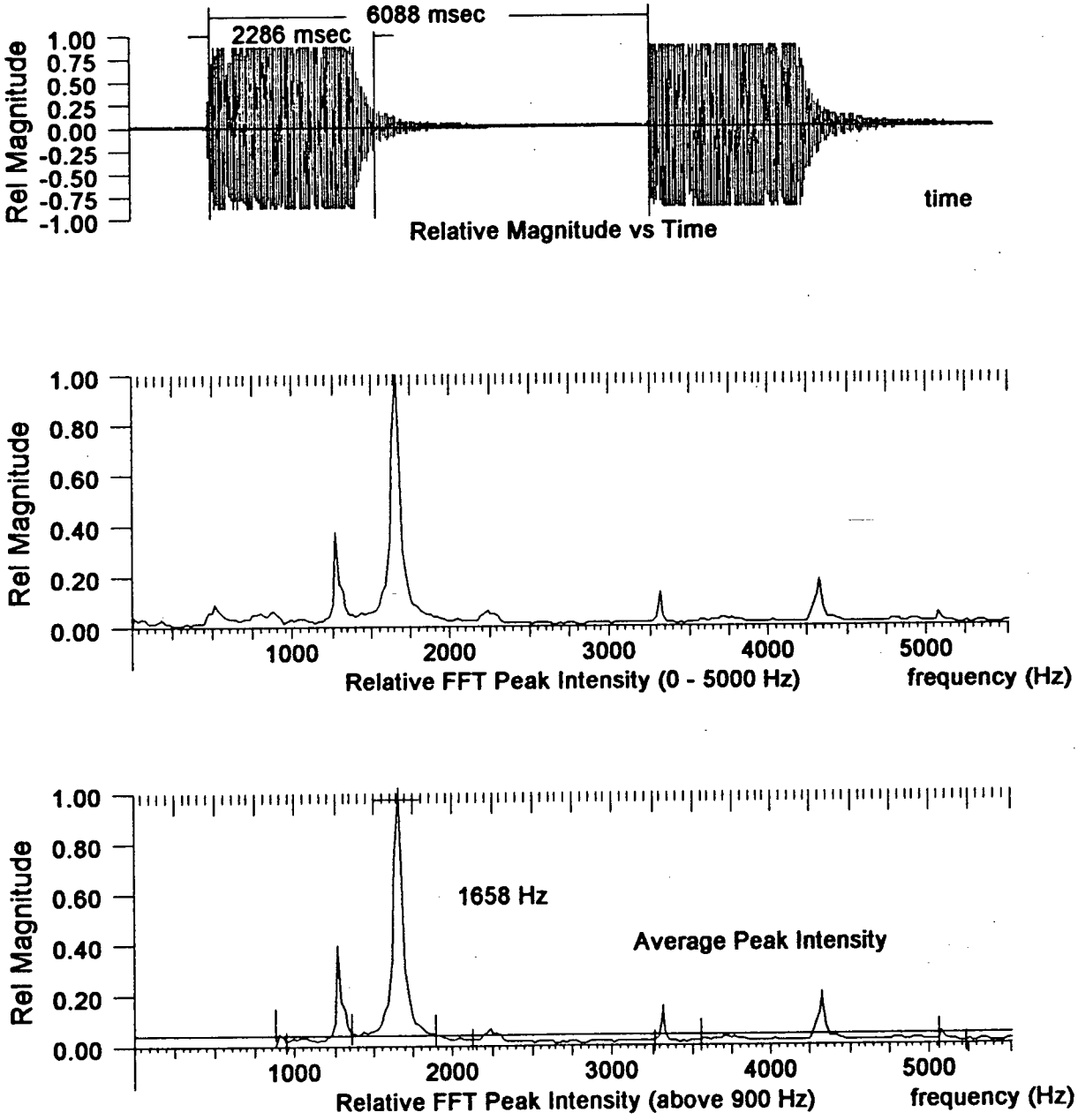
As can be seen from figure 5.3 (a) the majority of the spectral energy content in the warning signal is in the range above 900 Hz. Notice the narrowness of the characteristic peak and the lack of secondary peaks. This is common for warning signals that are generated electronically. Warning signals generated mechanically, such as the ring of a telephone bell, generally show a slightly broader characteristic peak as well as multiple harmonic peaks as shown in figure 5.3 (b). For the general background noise case shown in figure 5.3 (c), in this example FM rock music and voices, although there are secondary peaks at frequencies above 900 Hz, most of the energy content and the main peaks in the power spectrum are at frequencies well below the 900 Hz range.

The third graph of figure 5.3 (c) shows why, if we selectively look only at those peaks in the power spectrum above 900 Hz, we require the preset minimum limit for peak detection. When the low frequency energy content is eliminated from the peak detection process, the



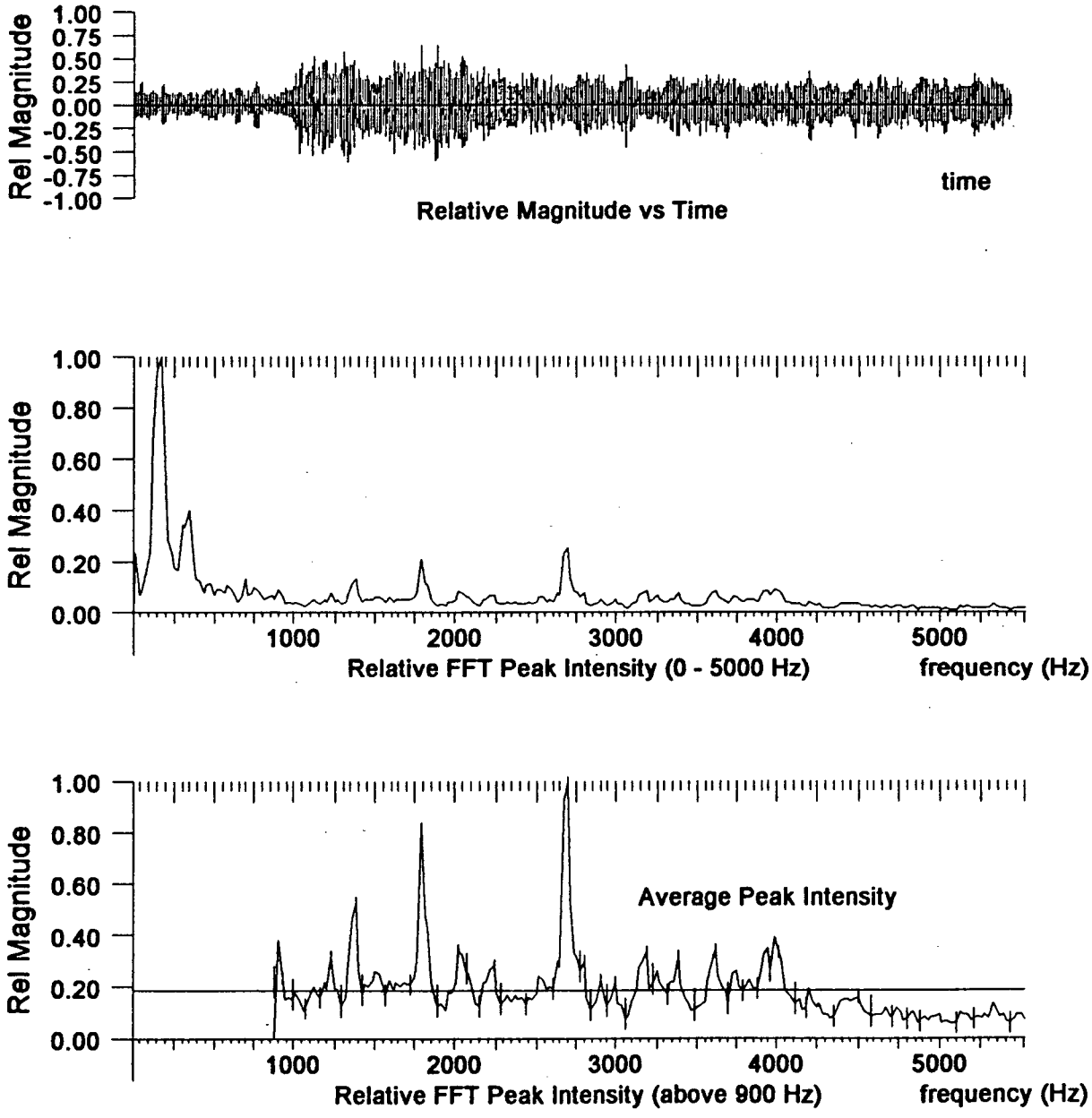
(a) Microwave Oven Alarm Spectral Analysis

Figure 5.3 - Typical Warning and Noise Signal Power Spectra



(b) Mechanical Telephone Ringer Spectral Analysis

Figure 5.3 - Typical Warning and Noise Signal Power Spectra



(c) Typical Background Noise (FM Rock Music & Voices) Spectral Analysis

Figure 5.3 - Typical Warning and Noise Signal Power Spectra

relatively minor noise peaks become dominant in the high frequency power spectrum. This could lead to erroneous spectral matches. These matches are not likely cause warnings to be issued to the user since a particular noise spectrum is unlikely to persist long enough to match the timing criteria of the timing analyser. However, by eliminating the noise peaks at this stage, we can increase the overall processing speed of the WARNSIS II algorithm.

In the recognition mode, if one of the detected peaks in the input signal's spectrum matches the characteristic peak of one of the learned signals in one of the templates, then a signal match is deemed to have been made for that template. For the recognition mode acquisition algorithm, as for the learning mode signal acquisition algorithm, two consecutive signal matches are required in order to determine if a "candidate" warning signal has been identified. A "candidate" signal is one for which further spectral and timing analyses shall be performed to determine if indeed an actual warning signal has been successfully detected. Unlike in the learning mode, however, in the recognition mode, a warning signal's candidacy is based on energy content at its characteristic frequencies and not on total energy (amplitude) increases. Once the "candidate" signal is associated with a given template, the start-of-burst indicator flag is set and the burst start time is recorded.

The WARNSIS II algorithm uses multiple templates, one for each learned signal, and each template has its own set of timing records and flags. Thus, it is possible for WARNSIS II to identify multiple warning signals overlapping in time. Provided that the characteristic peak of each of the different warning signals in the input to WARNSIS II are at a different frequencies, and that each of the warning signals in the input possesses enough energy to meet the minimum energy (magnitude) criteria stated earlier, the peak for each individual warning signal in the combined signal will be detected by the algorithm.

Once a warning signal's candidacy has been determined, that is a spectral template match has been found, the burst duration timer is kept running until the warning signal's peak is no longer detected (lost). In the recognition mode, losing the signal requires two consecutive absences of the given warning signal's characteristic peak in the input signal. Once the signal is lost, the burst duration is calculated as it was in the learning mode, from the time of detection of the *first* of the two initial matches between the input signal's spectral peaks and the warning signals characteristic peak stored in the template to the time of the *first* of the two consecutive peak losses. At this point a determination can be made as to whether a spectral and burst timing match can be made between the input signal and a previously learned signal whose characteristics are saved in the template. A "match" between an incoming signal and a template requires both the spectral characteristics and the burst duration timing of the input signal match the corresponding template values within pre-defined limits. In particular, the burst duration comparison algorithm allows a burst duration match to fall within a time window of variable length. From examining the variation in a number of real input signals, it was determined that the input signal's burst duration need only match the template values within +/- 10%. This means that the detected burst can be up to ten percent longer or ten percent shorter than the value stored in the template. Similarly, the spectral peak comparison algorithm was determined to require a tolerance of +/- 1 frequency bin (+/- 15.6 Hz for 512 sample intervals @ 8012 HZ) in order to successfully match a peak from the input signal to a characteristic peak in a template. Both tolerances can be defined on either a template to template basis or as an overall tolerance limit for WARNSIS II and, as for all other WARNSIS II limits, can be changed on an implementation to implementation basis. If both the timing and spectral characteristics have been matched between the input signal and one of the learned warning signal templates, a burst recognition flag is set indicating a valid burst match for that warning signal.

For continuous signals, the burst duration must only be longer than the minimum burst duration that is stored in the template, rather than actual burst duration. If WARNSIS II is trying to recognize a continuous signal, once the minimum burst duration is exceeded a continuous recognition flag is set. For a continuous signal, the recognition flag can be maintained for the duration of the signal, or it can be cancelled automatically after some period of time and re-issued if the warning signal remains on long enough to exceed the minimum burst duration a second or third time.

WARNSIS II also has the ability to further validate repetitive signals such as telephone rings by waiting for a second occurrence of the warning signal to appear. When in the learning mode, we assume that the second increase in amplitude indicates a second signal burst. In the recognition mode, WARNSIS II requires a second complete burst to occur before it can attempt to match the interval times of the incoming signal with that of the template's. Rather than just waiting for a second burst to start, the entire second burst must occur and match the same timing and spectral characteristics as the first. This means that noise bursts, as well as intervening signals from other valid warning devices will not affect the interval timer. The interval timer also allows a +/- 10% variation in interval time between the input signal and the warning signals template value for a valid match to occur. If an interval timing match is found an interval recognition flag is set. The use of the various recognition flags will depend on the implementation of WARNSIS II in an actual device. In some cases a notification to the user that a warning signal has been detected could be issued, in others the recognition flag may be used to trigger further processes in an existing medical device.

Figure 5.4 shows the flowchart for the generalized WARNSIS II recognition algorithm. The DSP code for both the learning and recognition modes can be found in Appendix A.

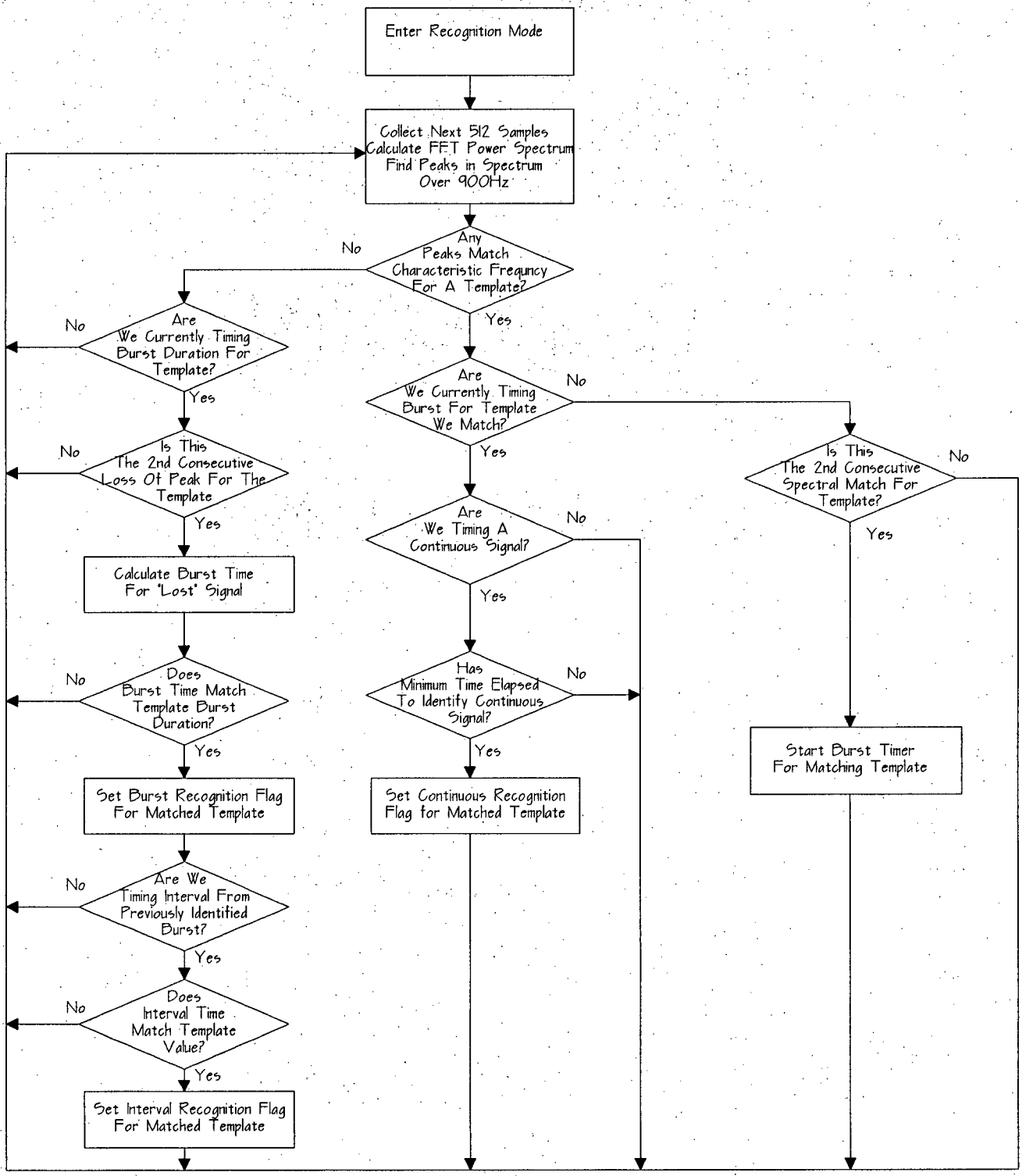


Figure 5.4 - Flowchart for WARNSIS II Recognition Mode algorithm

Chapter 6

6.0 WARNSIS II Implementation and Evaluation

The WARNSIS II project has resulted in two separate implementations of the WARNSIS II algorithms. Much of the initial design and testing was performed on a PC as a MS-Windows compatible program. This allowed for rapid modification and testing of the algorithms as well as providing a rich graphical environment for displaying and analyzing results. Once the basic algorithms had been designed, the system was then implemented on a Texas Instruments TMS320C30 Digital Signal Processing Evaluation Board. This provided the platform for testing the algorithms in real-time on a lower cost system. Finally the two implementations were integrated to allow PC control of the algorithms operating on the DSP board.

6.1 PC Based System

The PC version of the WARNSIS II algorithms were written in Turbo Pascal for Windows Version 1.5 to run under Microsoft Windows on any IBM compatible PC system. In addition to performing the WARNSIS II algorithms, the PC based program also provided a number of additional routines that were used to display and analyse digitized input signals. This was very useful in the early stage of development and lead directly to the development of a number of the WARNSIS II algorithms. By visually examining the digitized signals and their Fourier transforms, it was possible to determine what features made the warning signals distinct from other types of input signals. Indeed, many of the WARNSIS II algorithms are implementations of the pattern recognition "routines", such as picking out distinct peaks from the background, that were originally performed visually.

The PC based system also provides a "toolbox" environment for testing and refinement of new algorithms in the future. As new types of recognition algorithms are designed they can easily be added in a modular fashion to the existing PC program and then tested. This helps to meet one of the objectives of the WARNSIS II project, namely to provide a method for easily maintaining and upgrading the WARNSIS II system.

To allow the maximum flexibility of the PC-based system, again rather than relying on an implementation specific format, digitized signals are input to the WARNSIS II program in VOC format. This is a standard PC format originally created by Creative Labs for their "Sound Blaster" series of signal processing boards. This allows any PC based sound acquisition board to function as the input system for a PC-based WARNSIS II system. For this portion of the project a Creative Labs Sound Blaster_{16} was used to provide the digitized input signals.

Using the standard VOC file format and a non-specific input device also provides two other useful features for WARNSIS II development. Since the VOC format provides for the storage of input signals in digital format as a file, new algorithms can be tested against existing digitized signals. This means that comparison of the new algorithms can be made against existing algorithms on the same signal. Even the best quality tape recordings will deteriorate over time whereas the digitized files can be used over and over. The second advantage comes from the ability to use VOC files from any source. If a test of a specific warning signal is required, a VOC file for the warning signal can be easily sent via modem or network from a remote location.

6.1.1 Results of PC Based Implementation

Figure 6.1 (a) and (b) show an example of the results of the WARNSIS II learning mode algorithms. The top graph shows the input signal amplitude versus time for approximately 6.33 seconds of telephone ringing signal generated by an electronic, rather than mechanical bell, ringer. The entire first burst of the telephone ringing signal is shown, but only the beginning of the second ringing burst is shown as learning mode does not require the entire second burst for interval determination. Additionally, the top graph shows vertical lines indicating the beginning and end of the first burst and the beginning of the second burst calculated by the STAAA method. The burst duration and interval times calculated using the STAAA method for this signal were found to be 2.189 seconds and 6.178 seconds respectively which is in agreement with the telephone standards [29].

The middle graph shows the overall FFT power spectrum for the input signal for the time period between the vertical lines indicating the start and end of the first ringing burst. As stated earlier, this is the combined power spectrum totalling each of the FFT magnitudes calculated for the 46 individual 512 sample FFTs performed during the burst. The bottom graph shows the combined power spectrum for frequencies above 900 Hz and also shows the peaks in the power spectrum identified by learning mode. The small vertical lines at the bottom of each peak show the width of the peak as detected by the peak detection algorithm as well as the centre frequency for each peak, shown by the small vertical line at the top of the peak. The characteristic peak is highlighted at the top of the third graph by a horizontal line showing the width of the characteristic peak. Although, at present, the width of the characteristic peak is not used in recognition mode (nor calculated in the DSP version), it is included as a feature in the PC version of the learning mode in order to supply more spectral information when

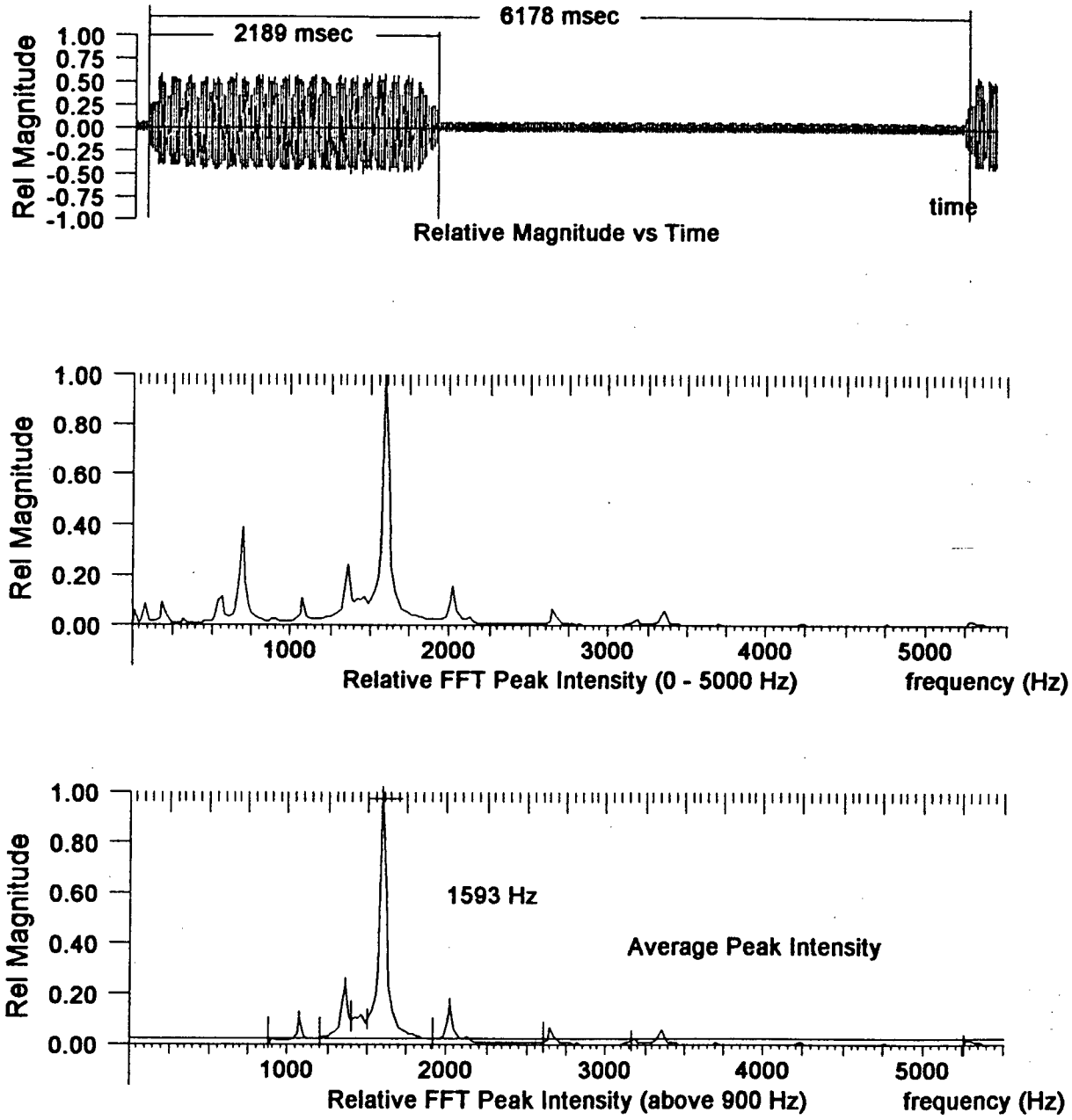


Figure 6.1 (a)- Results of Learning Mode for Electronic Telephone Ringing Signal

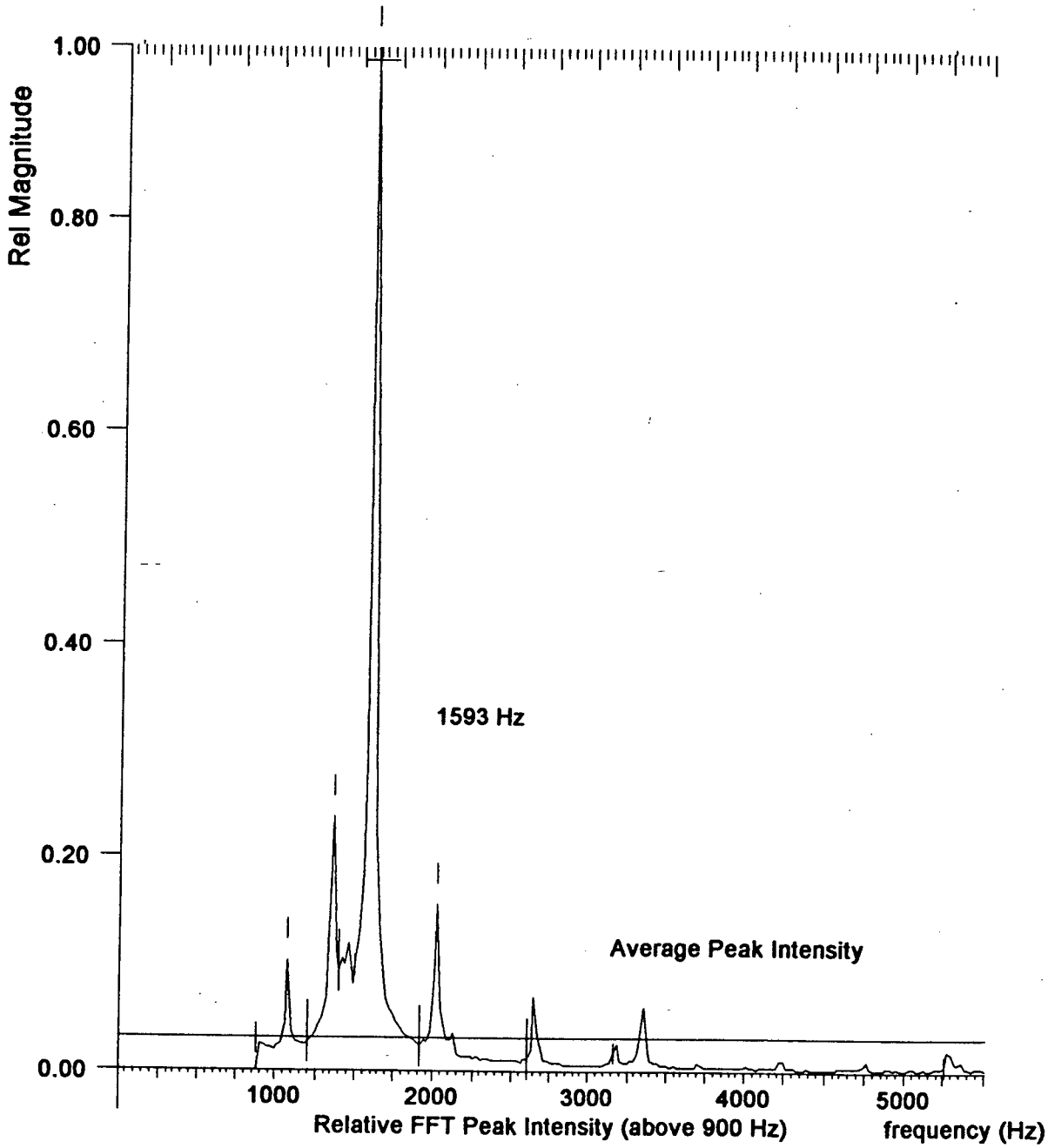


Figure 6.1 (b) - Results of Learning Mode Electronic Telephone Ringing

developing new algorithms. For the telephone ringing signal shown in this set of graphs, the characteristic peak frequency was determined to be 1,593 Hz with a width of 275 Hz from 1,458 Hz to 1733 Hz. The centre frequency is in close agreement with the frequency of 1575 Hz measured by oscilloscope and is within the accuracy limits for the FFT at the sample rate of 11,025 Hz and 512 samples. The horizontal line near the bottom of this graph shows the average peak power above 900 Hz and again is shown as a development "tool" rather than as one of the parameters for signal recognition.

Figure 6.2 (a) shows graphically the overall results obtained by the WARNSIS II recognition mode algorithms from a complex signal. The graph in Figure 6.2 (a) shows 7.64 seconds of digitized signal. The signal is a composite of four separate signals:

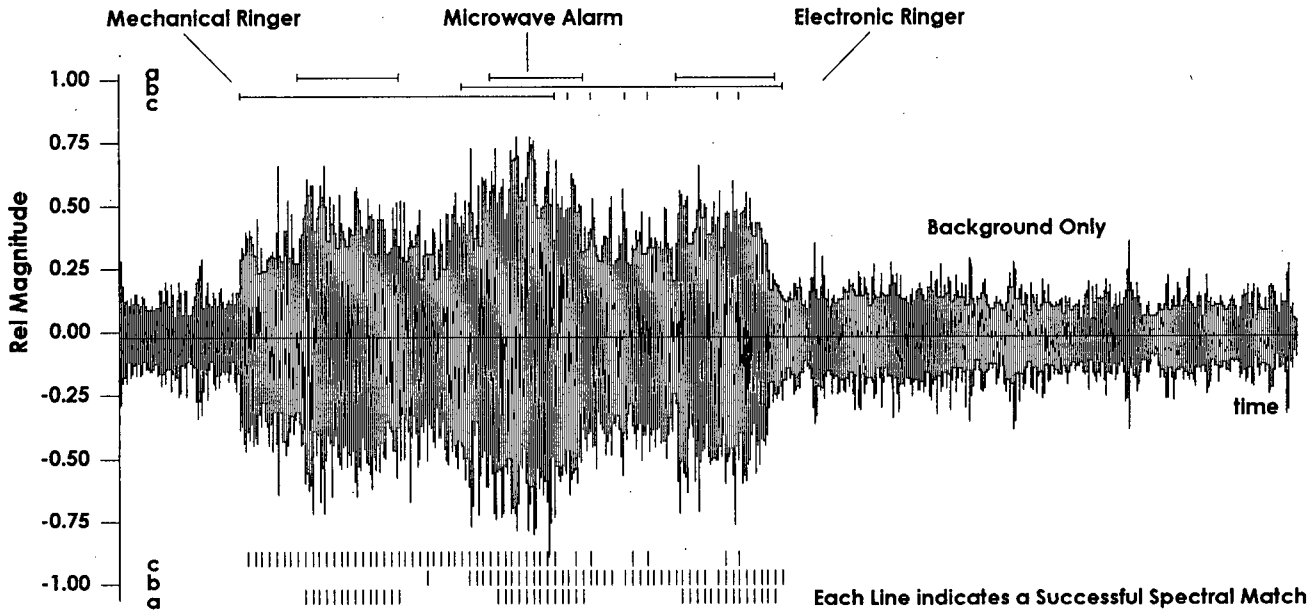
i) Background Noise (FM rock radio and voices)

ii) Warning Signal A, microwave oven alarm

iii) Warning Signal B, electronic tone telephone ringer

iv) Warning Signal C, mechanical bell telephone ringer

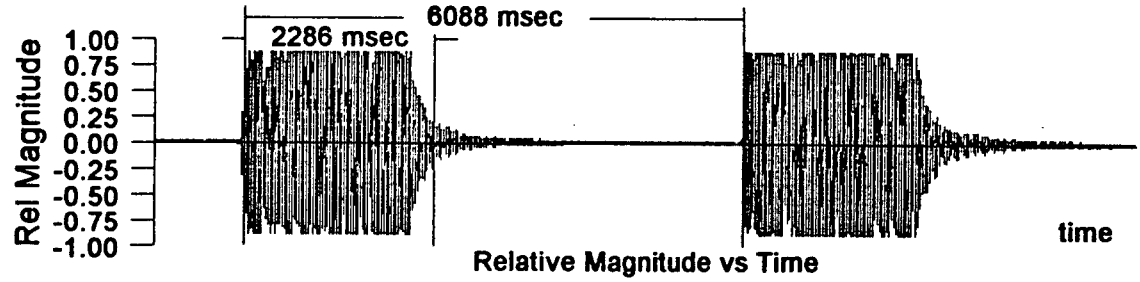
Figure 6.2 (b), (c) and (d) shows the individual digitized warning signals without the background noise signal. For each of the warning signals in Figure 6.2 (b), (c) and (d) the values showing the characteristic peak frequency, burst duration time and interval time were obtained from the WARNSIS II learning mode algorithm. Figures 6.2 (b) and (c) show multiple bursts so that interval times can be shown, whereas the composite signal shown in. Figure 6.2 (a) contains only a single repetition of signals B and C.

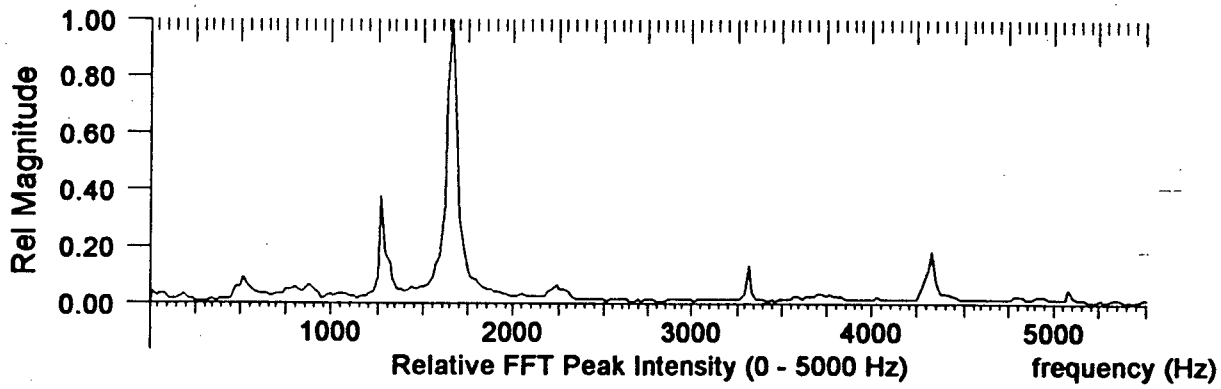


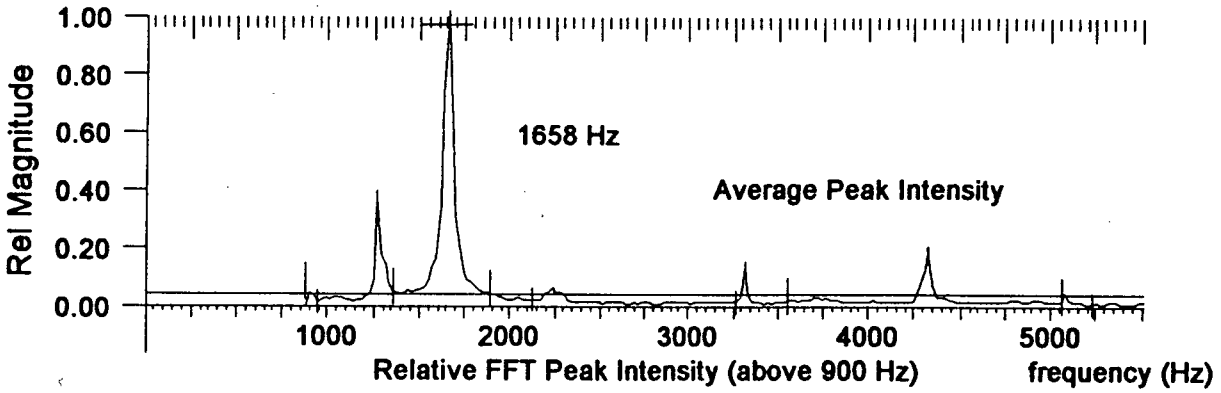
Burst Duration:	Interval:	
1: 650 ms (microwav.wtp)		Dur. Match : microwav.wtp
2: 2043 ms (mech_phn.wtp)		Dur. Match : mech_phn.wtp
3: 604 ms (microwav.wtp)	1254 ms	Dur. Match : microwav.wtp Int. Match : microwav.wtp
4: 139 ms (mech_phn.wtp)		
5: 139 ms (mech_phn.wtp)		
6: 139 ms (mech_phn.wtp)		
7: 650 ms (microwav.wtp)	1207 ms	Dur. Match : microwav.wtp Int. Match : microwav.wtp
8: 2090 ms (elec_phn.wtp)		Dur. Match : elec_phn.wtp

(a) - Results of WARNSIS II Analysis of Composite Signal

Figure 6.2 - Composite and Individual Warning Signals Analyses. The figure above shows the results of the PC-based recognition mode analysis. The central graph of the figure shows a combined signal consisting of background noise and three different warning signals: a - Microwave Oven Alarm, b- Electronic Telephone Ringer, c - Mechanical Telephone Ringer. The set of horizontal lines at the top of the graph, indicated as a, b and c, show the position of each of the individual warning signals in the composite signal , as determined by the recognition mode analysis. The small vertical lines at the bottom of the graph, indicated as c, b and a, show each successful spectral match for each warning signal in the composite. Each vertical line corresponds to one 512 sample period. Each of the horizontal lines above the graph corresponds to one set of vertical lines below the graph are arranged symmetrically around the central time axis.

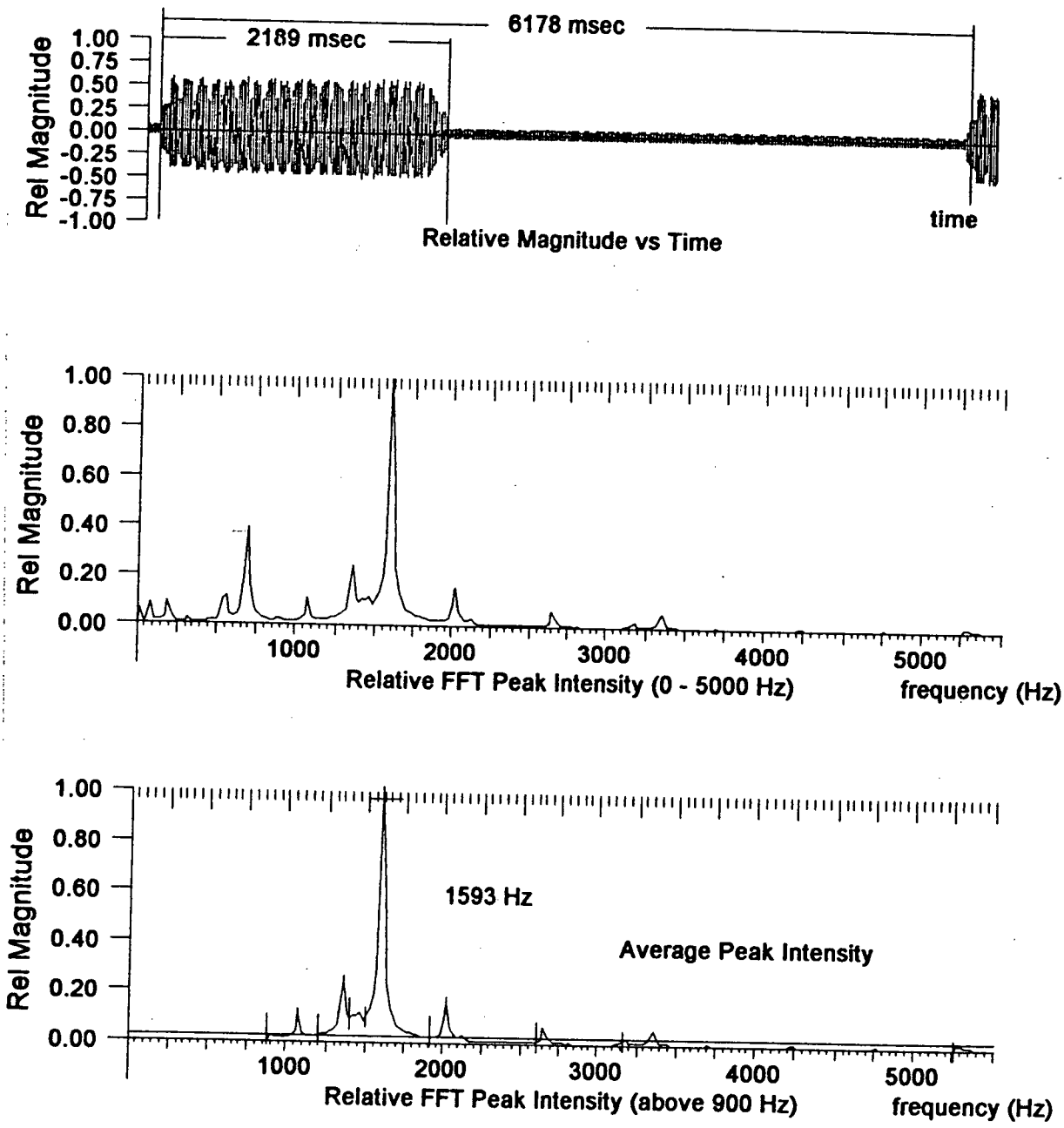






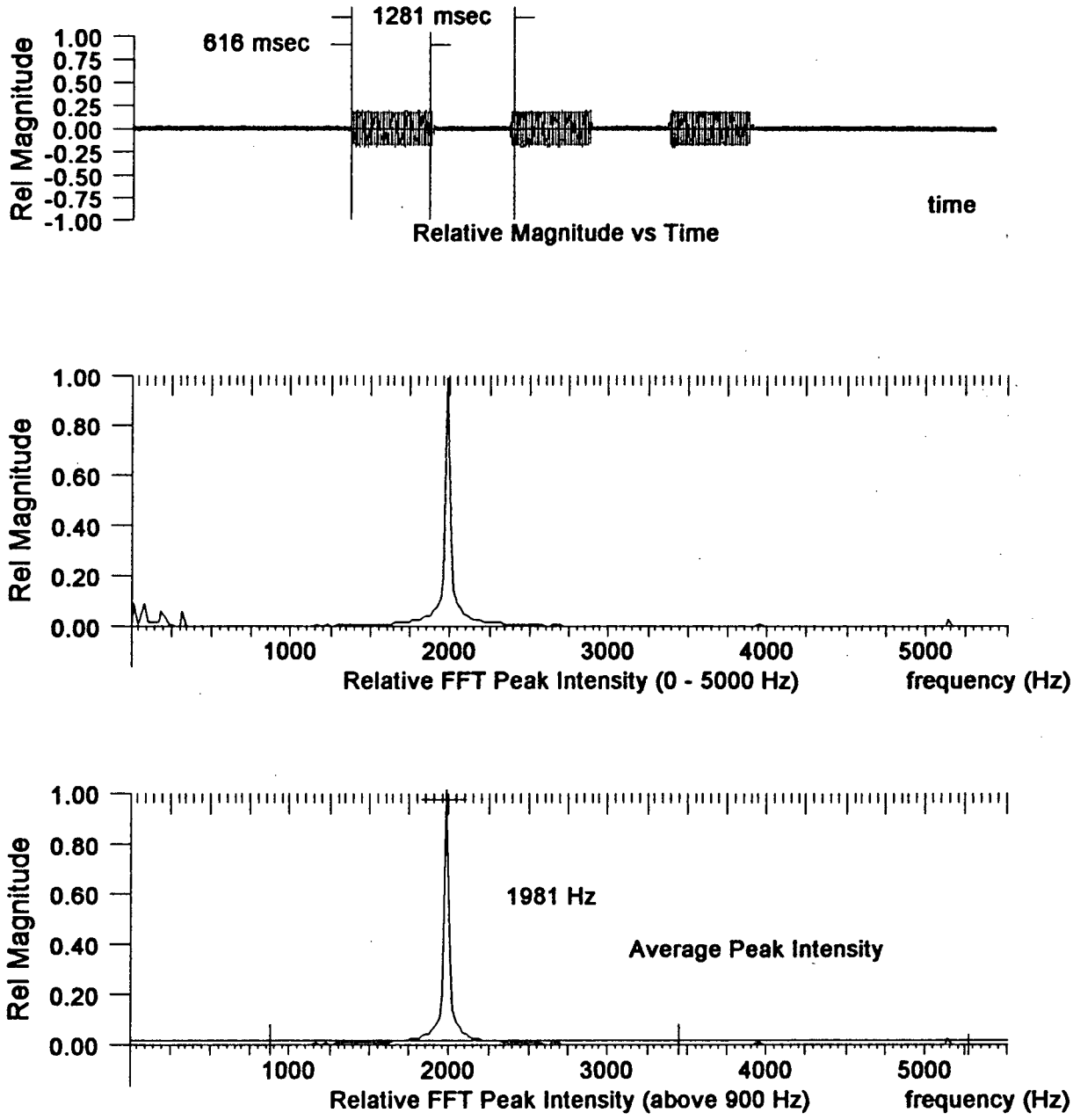
(b) - Mechanical Telephone Ringer Signal and Spectral Analysis Signal C

Figure 6.2 - Composite and Individual Warning Signals Analyses



(c) -Electronic Telephone Ringer Signal and Spectral Analysis Signal B

Figure 6.2 - Composite and Individual Warning Signals Analyses



(d) - Microwave Oven Alarm Signal and Spectral Analysis. Signal A

Figure 6.2 - Composite and Individual Warning Signals Analyses

Figure 6.2 (a), produced directly by the PC based implementation, shows the combined results of the timing and spectral analysis algorithms. For each warning signal in the composite signal, the spectral matches between the peaks in the FFT power spectrum for the composite signal and the characteristic peaks stored in the templates for each of the warning signals is shown by a small vertical line below the digitized signal. Each vertical line corresponds to a spectral match for one 512 sample time interval. The spacing between each vertical line is not constant due to the resolution limits of the display screen, but the timing is identical for each of the 512 sample intervals. The timing analysis for each warning signal burst is shown by the horizontal lines above the digitized signal. The corresponding spectral and timing indication lines are symmetric about the horizontal axis at the centre of the digitized signal, shown by labels a-a, b-b etc.. Thus the first set of spectral match lines below the digitized signal correspond to the first horizontal timing line above the digitized signal.

Below the graph, the calculated burst duration and interval times as well as burst duration and interval match messages are shown for the warning signals in order of their recognition. Thus, although warning signal C started before warning signal A, the burst for signal A ended before that for signal C, allowing A's burst duration to be calculated and the user message for signal A to be generated before signal C ended.

The spectral matching acquisition algorithm's requirement for two consecutive matches or losses can be clearly seen in the spectral match lines for warning signal B. Approximately half way into signal C's spectral match series a single spectral match for signal B is shown on the graph. Since this was a single occurrence, the two consecutive spectral match criteria was not met and the signal's candidacy was not successful at this stage. Approximately 6 time segments later signal B was successfully determined to be a candidate. The signal remains a

candidate until the end of the warning signal burst, even though there are two occasions indicated by missing vertical lines when a spectral match for a single 512 sample interval is not found for signal B. This is one of the key features of the spectral analysis algorithm. It allows the temporary loss of the signal without interrupting the timing analysis.

The two consecutive loss algorithm can be extended to three or more consecutive losses for determining candidacy or loss of a signal. This could allow the WARNSIS II signal recognition algorithm to operate in a higher noise environment than for the standard two match/loss algorithm. For example, for the two consecutive loss algorithm, a signal would be lost after being masked by background noise for only two time periods. If a signal requires three or four consecutive losses to end the signal's candidacy, then in a very noisy environment the signal could be masked by background noise for up to four time periods without being lost. This would allow noise bursts to be longer in duration without interrupting the timing analyser.

This would also, however, have the effect of lengthening the time after the warning signal ends before the burst recognition flag is set, since three or more time intervals, instead of two, would be required to elapse after the end of the signal burst. This would not necessarily cause a problem as the time intervals are on the order of 64 msec (sample rate for the PC version of WARNSIS II was 11,025 Hz) and 4 time intervals would only amount to 1/4 of a second delay. A greater problem would occur from linking spurious noise bursts into a single long duration burst. The higher the number of dropouts allowed in the signal matching algorithm, the greater the likelihood that a series of noise burst would be grouped into a single burst. This could cause a false positive message to be issued, especially for continuous signals where a timing match only requires exceeding a minimum burst duration time.

Figure 6.2 (a) also shows why a two part spectral and timing analysis is required. Although the WARNSIS II spectral analysis correctly identifies the spectral matches for the initial occurrence of warning signal C, after warning signal C ends, the spectral matching algorithm continues to find a number consecutive spectral matches between the input signal and the characteristic peak for warning signal C. These are listed in the burst duration calculation tables shown below the graph as 139 ms duration bursts. However, because these bursts do not match the timing requirements of the timing analysis algorithm, no burst duration or interval time match message is issued to the user. If only spectral information had been used, the recognition of the signal by the spectral analyser alone would have issued an erroneous message.

As can be seen in Figure 6.2 (a), the use of both spectral and timing analysis in combination with the use of multiple templates (one for each warning signal learned) allows the WARNSIS II system to detect multiple overlapping warning signals in a noisy environment. Figure 6.2 (a) shows at one point the simultaneous occurrence of the three warning signals in addition to the background noise. It should also be noted that the characteristic peaks of warning signals B and C are at frequencies of 1658 Hz and 1593 Hz respectively. This is a separation of only 65 Hz or 3 *bins* for a sample rate of 11,025 Hz. Since the spectral matching algorithm allows for a variation of plus or minus one bin for a valid spectral match, this is very close to the operational frequency resolution limits of WARNSIS II.

6.1.2 Other Results

As the spectral analysis for WARNSIS II is performed in software by the using the FFT, it is important to determine the effects of digitization, sampling rate, and the data sampling

window on our algorithm. The PC-based implementation provides not only a basis for implementing the WARNSIS II algorithms but also for testing the effects of various parameters and features of the WARNSIS II algorithms. The following sections detail the results of those investigations and their effects on the WARNSIS II algorithm.

6.1.2.1 8 Bit vs 16 Bit Digitization

The VOC format provides for the storage of digitized signals in both 8 and 16 bit data sizes. Initially both 8 and 16 bit data formats were used for digitizing the input signal when testing the WARNSIS II algorithms. An 8 bit digitization allows for 256 levels of input signal (+/- ~ 0.4%), a 16 bit digitization allows for 65536 levels (+/- ~0.002%). Calculating the STAAA's averages the input signal's magnitudes over a time period of 512 samples. If we assume a maximum error of ½ of a bit between the real and digitized input signals for each of the 512 samples then the maximum error for 8-bit digitization is 512*1/2 bit = 256 bits. For 512 samples this would give a maximum error of 256/512 = ½ bit maximum error after the average is taken. The result is the same for 16-bit digitization. Since the STAAA process requires a increase of 50% in the input signal amplitude above the background level to start the learning process, the difference between 0.4% and 0.002% is irrelevant.

As the spectral analysis relies only on relative magnitude differences in the FFT power spectrum, the extra resolution provided by the greater number of input bits is of no extra value. Since peaks lower in magnitude than 25% of the largest peak found and lower than a preset minimum are ignored, the extra resolution is lost in the background. Given these results, it was found that when performing either the STAAA analysis or the FFT analysis that the data digitization at either 8 or 16 bits had no effect on the function of the WARNSIS II algorithms.

6.1.2.2 FFT Size, Frequency Resolution and Sampling Rate

For the FFT the relationship between frequency resolution, sampling rate and sample size is given by,

$$\Delta f = R / N \qquad (6.1)$$

where Δf is the frequency resolution between adjacent bins, R is the sampling rate and N is the size of the FFT. The larger N, the FFT size, becomes, the smaller Δf becomes. However, the larger N becomes, the longer the period of time a single FFT power spectrum covers. These are the time-frequency resolution limits described earlier in section 4.1.5.

There is no theoretical limit on sampling rate and size, and time-frequency resolution for WARNSIS II. The WARNSIS II algorithms can be implemented at any sampling rate. The trade-off, again is between the time and frequency resolution. For a WARNSIS II device to operate successfully it must be able to meet the two following criteria. Firstly, to discriminate between multiple overlapping signals, the characteristic peak frequencies of the two signals must be resolved separately from each other. In order to do this the frequency resolution of the FFT must be fine enough to place the characteristic peaks from two separate signals into different *bins*. Secondly, if two warning signals have identical characteristic frequencies, then they can only be successfully recognized if they do not overlap in time, and they have different burst and/or interval times.

This means that the time-frequency resolution issue must be solved on a practical basis such as the processor memory requirements or expected signal characteristics. If the sample

size was set so that only a few *bins* covered the entire spectrum, then the frequency performance of the WARNSIS II algorithm would suffer accordingly. Conversely, if the sample size is too great then the memory requirements and the time resolution would be similarly effected.

For the WARNSIS II algorithms, the size of 512 samples was chosen for two main reasons. First, choosing a sample size of 512 we can achieve a frequency resolution of approximately 21.5 Hz covering a period of 46.4 msec for a sampling rate of 11,025 Hz for the PC based implementation. This provides a reasonably good resolution for both frequency and time. Secondly, a sample size of 512 allows the DSP implementation to maintain two separate FFT sample arrays and processing code in the processor's internal memory. This greatly increases the speed of the DSP FFT algorithm [17]. By maintaining the same sample size for both implementations it is easier to compare algorithms between the two platforms.

More important than using a specific sample rate or sample size, is the requirement that the same sample rate and size be maintained both in the learning mode and in the recognition mode. Since not only the frequency resolution, but also the centre frequencies of the bins (hence the bin #) changes with sampling rate and size, a signal learned at one sample rate and size will not have the same centre frequency or bin number if the recognition algorithm operates at a different sample rate and size.

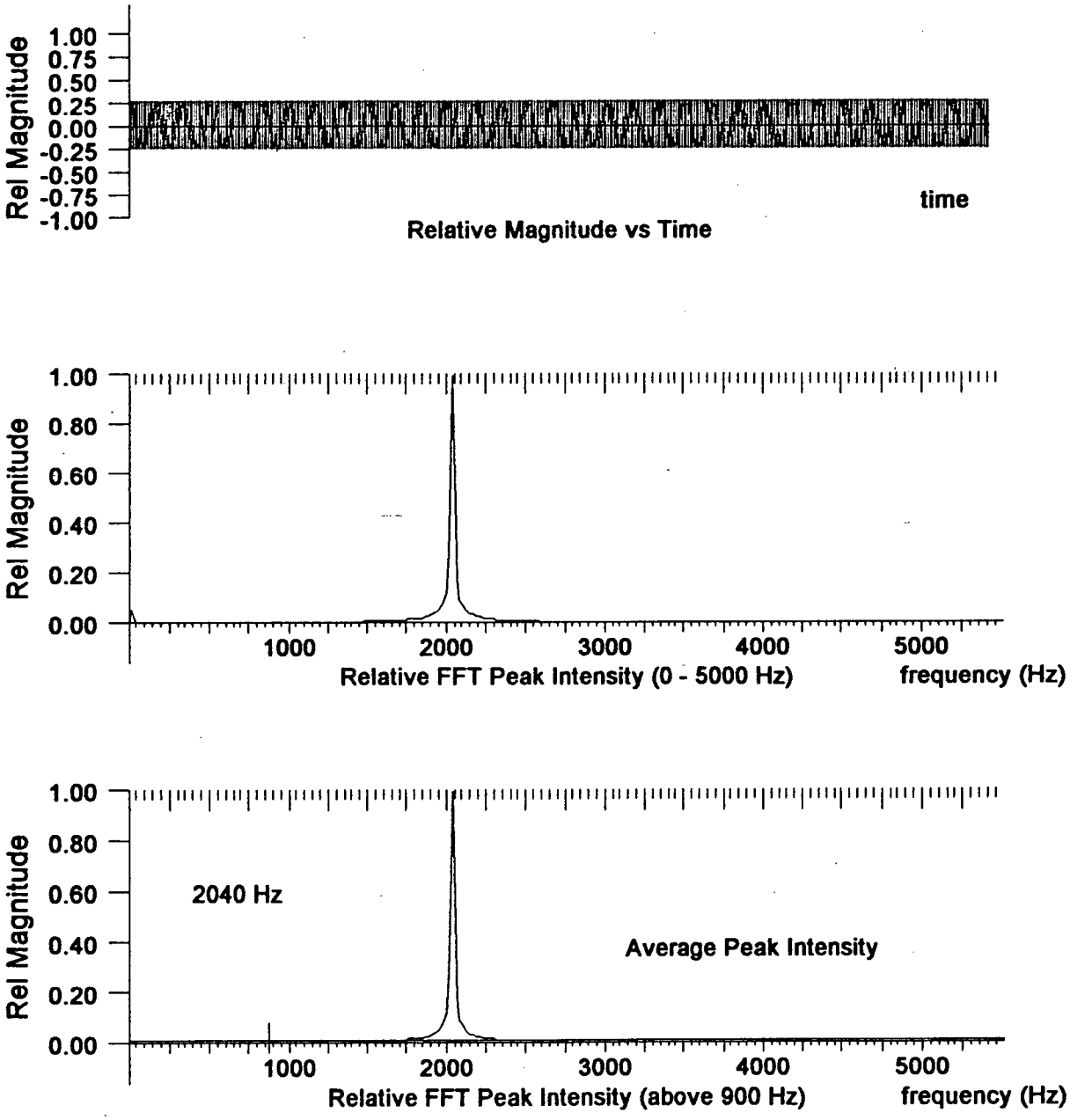
6.1.2.3 Sampling Window Effects

Section 4.1.1.2 discussed the effect that various sampling window functions would have on the leakage of energy between bins in the FFT power spectrum. Figures 6.3 (a), (b), (c)

and (d) show the effect of the standard square window, the Parzen window, the Welch Window and the Hanning window on the FFT spectrum of a 2040 Hz signal. Although the Parzen, Welch and Hanning window functions all improve the leakage between bins, as shown by the narrowing of the base of the peak, they provide little advantage for the WARNSIS II spectral analysis algorithm. The reduction in leakage is so small that it is lost in the background noise. This can be seen in Figures 6.4(a) and (b). Figure 6.4 (a) shows the power spectrum of the background noise signal using the standard square window. Figure 6.4 (b) shows the power spectrum for the same signal using the Hanning window function. There is virtually no difference in the spectrum aside from a slight "smoothing" effect that the Hanning window has on the peaks and valleys as was shown in Section 4.1.1.2. Since only the relative magnitudes of the peaks above the preset limit and above 25% of the maximum peak are important, the use of any one of the additional windowing functions simply adds to the processing time of the algorithm with no appreciable benefit.

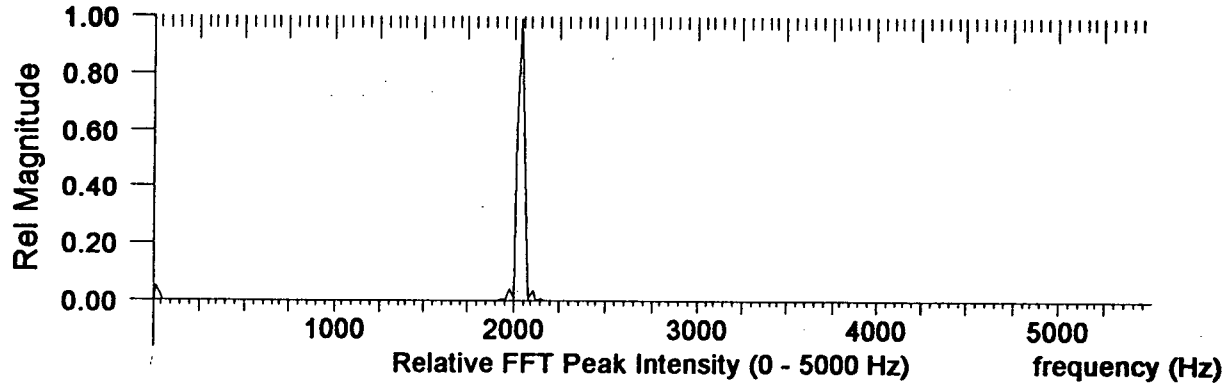
6.1.2.4 Speed Considerations

Although the PC implementation of the WARNSIS II algorithms operates on stored data files, an attempt was made to have the algorithms perform at a "real-time" rate. Due to the overhead of the Windows environment and the extra effort required by the PC's processor to display the graphical information generated by the program, there was not enough processing power to have the algorithms process all of the incoming data in real time. To speed up the processing time, rather than performing the FFT analysis on all of the data, the program was modified so that the FFT was only calculated for one of two, one of three, one of four, etc. sample periods. The time for the intervals for which the FFT was not calculated was included in the burst duration calculations, but no spectral analysis was performed on these missed

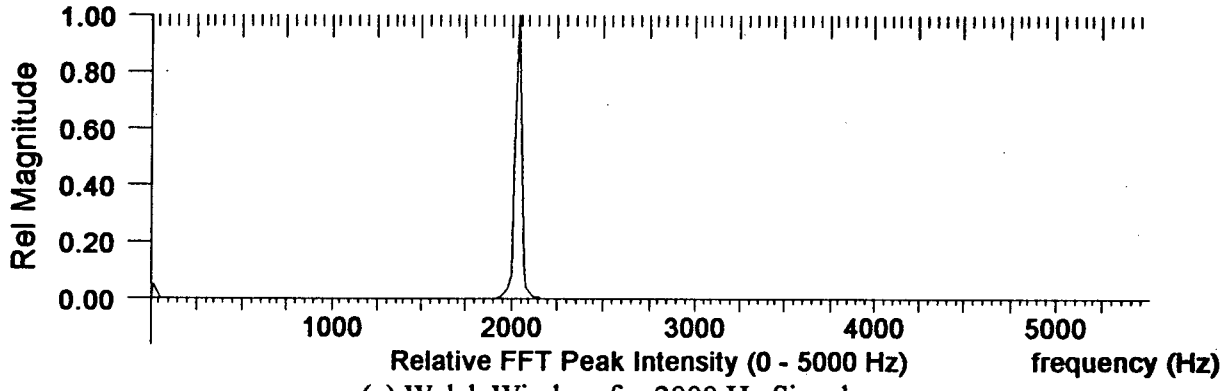


(a) Standard Square Sampling Window for 2000 Hz Signal

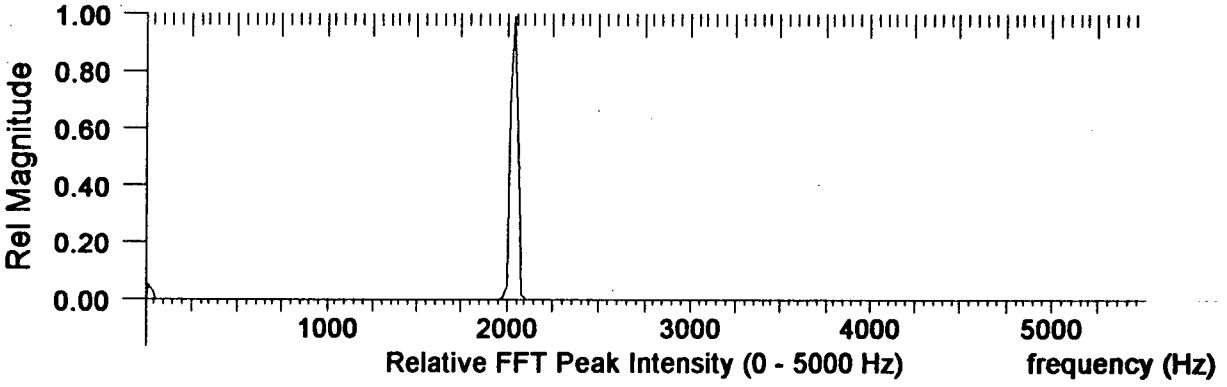
Figure 6.3 - Effect of Sampling Window Function on Power Leakage



(b) Parzen Window for 2000 Hz Signal

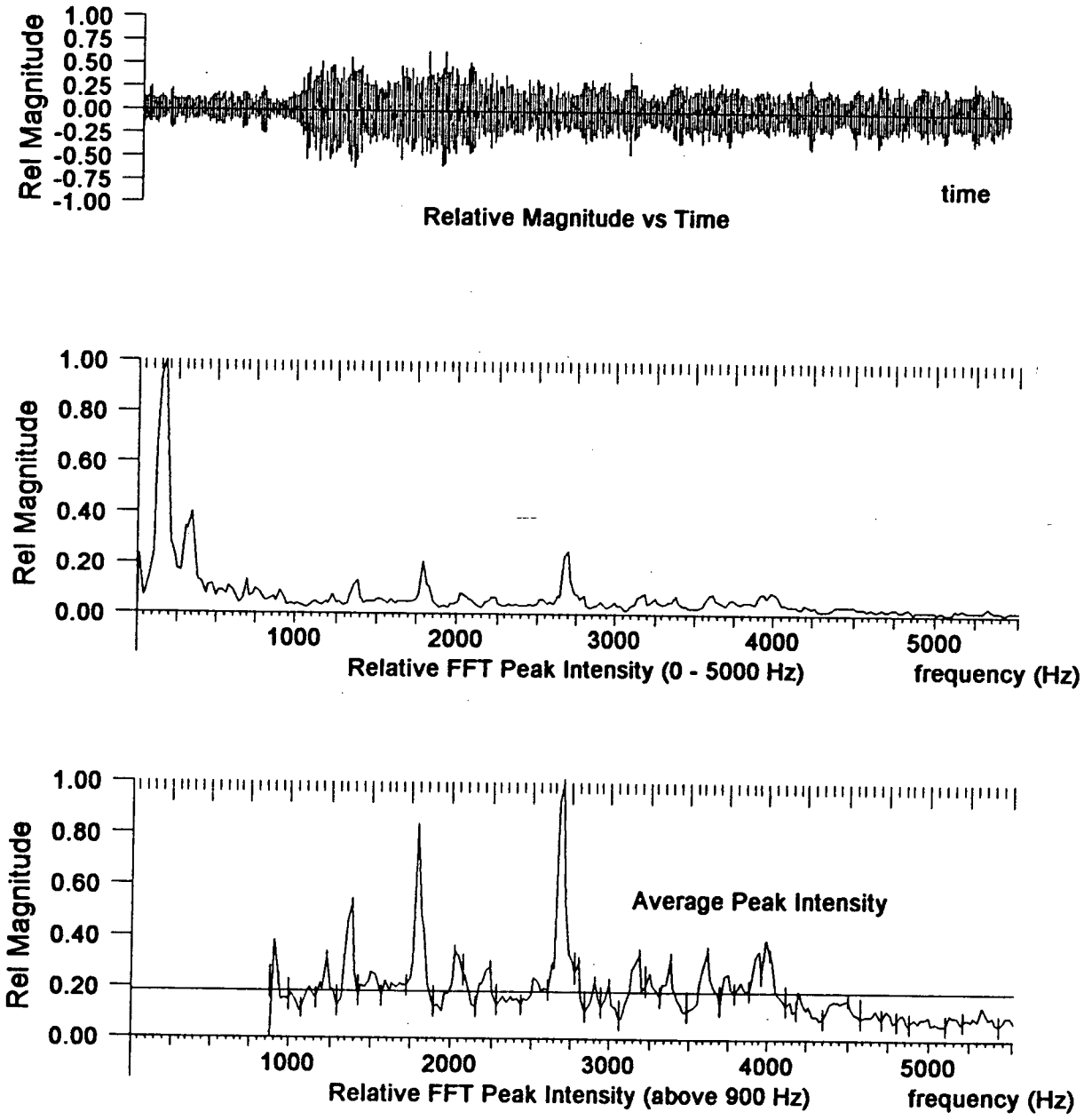


(c) Welch Window for 2000 Hz Signal



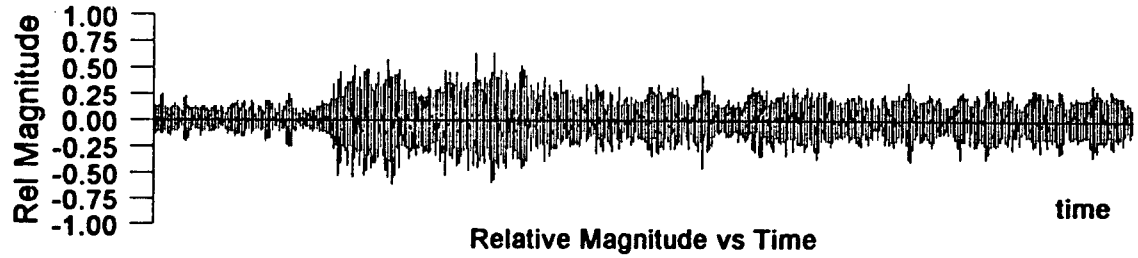
(d) Hanning Window for 2000 Hz Signal

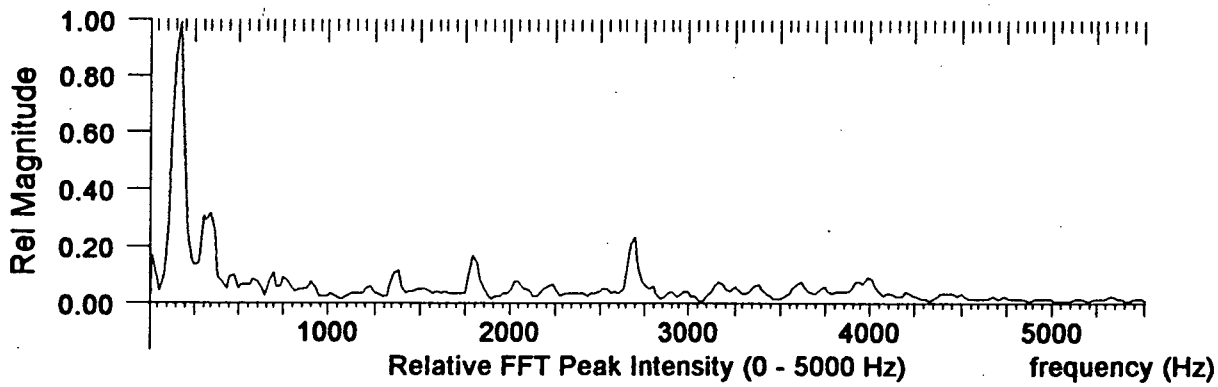
Figure 6.3 - Effect of Sampling Window Function on Power Leakage

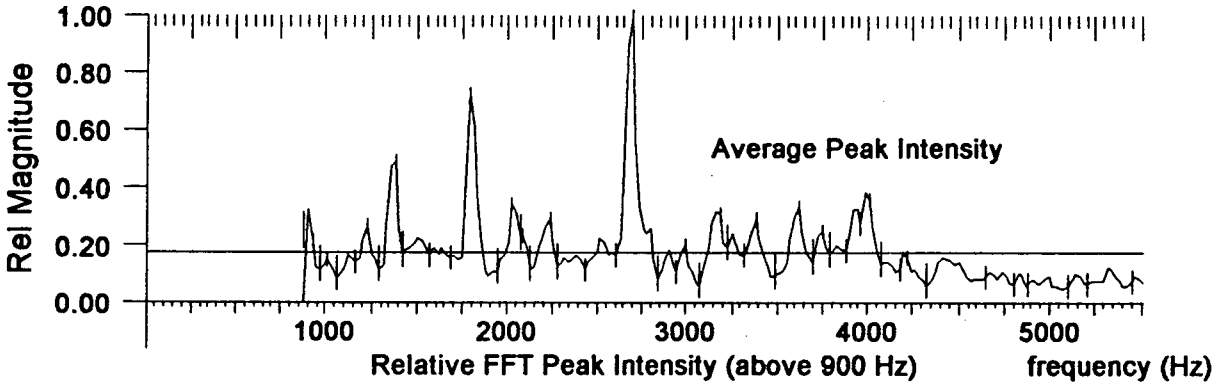


(a) - Background Spectrum for Standard Square Sampling Window

Figure 6.4 - Effects of Sampling Window, Background Noise Power Spectrum







(b) - Background Spectrum for Hanning Sampling Window

Figure 6.4 - Effects of Sampling Window, Background Noise Power Spectrum

intervals to determine if a signal was still a candidate or had been lost. The same two consecutive spectral matches or loss for signal acquisition or loss were still required, but these consecutive matches were now spaced further apart in time by the number of "missed" intervals. In comparison to those test signals that were successfully detected by the standard algorithm when no intervals were missed, the modified algorithms were able to miss up to 6 consecutive intervals and still successfully identify the warning signals. This brought the processing speed on a 50MHz 486 PC to the point where the signals could be processed in an amount of time less than or equal to the actual duration of the input signal .

An additional increase in processing speed was obtained by comparing the square of the peak magnitudes instead of the peak magnitudes. Since we are only interested in relative magnitudes and not absolute magnitudes there is no need to perform the square root function required to obtain the absolute magnitude from the real and imaginary parts of the FFT. This provides a great increase in speed for the spectral analysis algorithm since processing the square roots requires as much processing time as performing the FFT itself.

6.2 TMS320C30 Based System

The DSP version of the WARNSIS II algorithms were written in TMS320C3x assembly language code and implemented on a Texas Instruments TMS320C3x Evaluation Module (EVM). The EVM is a half length board that installs in an IBM compatible PC. The EVM can run in stand alone mode operating completely independently from the PC or can be interfaced to the PC through the PC's I/O port structure. The EVM itself consists of a 33 MFLOP 32 bit TMS320C30 Digital Signal Processor with 2K words of internal RAM. 16K words of additional zero wait-state RAM are also provided on the circuit board as well as a TLC32044

Analog Interface Circuit (AIC). The AIC provides 14 bit digitization for audio signals which can be input directly from a microphone or via a preamplified line [30].

In the WARNSIS II setup the AIC of the EVM was connected via the Creative Labs Sound $Blaster_{16}$ to a standard 500 Ohm unidirectional microphone. For the DSP implementation of WARNSIS II, the Sound $Blaster_{16}$ was used only as a preamplifier for the microphone and no digitization or filtering was performed by this board. The TLC32044 AIC however has an internal bandpass filter with the ripple bandwidth and 3-dB low frequency roll-off points of the highpass section at 150 Hz and 100 Hz respectively, and a high frequency 3-dB roll-off at approximately 3,700 Hz for a sample rate of 8,012 Hz [31]. The sample rate of 8,012 Hz is used for the DSP implementation as the AIC filtering circuits are optimized for this frequency. The lower sample rate for the DSP implementation vs the PC implementation presents no problems in that it still provides a maximum frequency resolution of 15.64 Hz and an interval time of 63.9 msec. The maximum frequency limit from the lowpass section of the AIC filter also presents no problems as all of the warning signals that we have identified so far have their characteristic peak frequencies well below this limit. As for the PC implementation, the actual choice of sampling rate and sample size is a practical one guided by memory requirements and desired frequency resolution, not a theoretical one.

The TMS320C3x Digital Signal Processor family was chosen due to the support available from the manufacturer as well as the availability of a number of versions of the processor at various speed and price ranges. Although the EVM cost was on the order of $1200 US, some versions of the processor itself can be purchased in quantity for under $20 US. This means that a relatively low cost implementation of a stand alone device based on the TMS320C3x DSP could be designed.

6.2.1 Program Size

The size of the compiled assembly language program for the DSP implementation was 1,492 words and required 1,801 words of data storage. Of the 1,492 words of program code, 952 words were required for implementing the WARNSIS II algorithms and communications routines with the PC, 420 words were required for the FFT routine, 78 words were required to implement a floating point divide routine and the remaining 42 words were for miscellaneous interrupt vectors and processor control. Of the 1,801 words of data storage, 1,024 were required for maintaining two 512 word tables, one for collecting the current 512 input samples, the second for simultaneously calculating the FFT of the previous 512 samples. The rest of the data storage was comprised of 256 words of storage for the FFT sums for learning mode, 256 words for the FFT "twiddle factors", 120 words for peak identification, 40 words for templates, 10 words for I/O buffers for PC communications and 55 words for miscellaneous variables. The majority of both code and data were accommodated in the 2K of internal RAM provided on the TMS320C30 DSP chip, meaning less than 2K of external RAM was required.

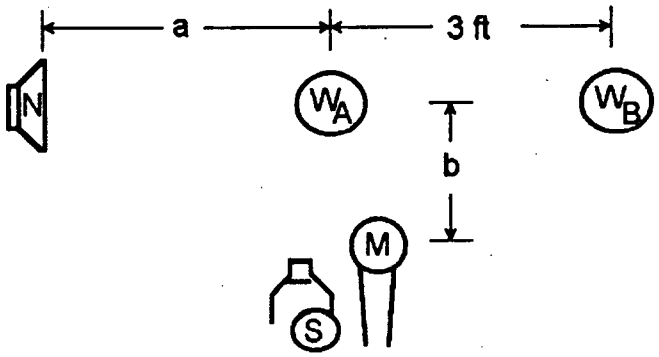
6.2.2 Results of DSP Based Implementation

The WARNSIS II spectral and timing analysis algorithms functioned similarly in the DSP based implementation as they had for the PC based implementation. The TMS320C30, however, provided enough processing speed to perform the analysis in real time. In fact, of the 63.9 msec interval time for 512 samples, approximately 0.57 msec was required to process the FFT from the previous 512 samples. As the current samples are input via an interrupt routine the time not used for processing the FFT's is available for the peak detection, spectral and timing analysis and template comparison algorithms. Also, because the data is input via an

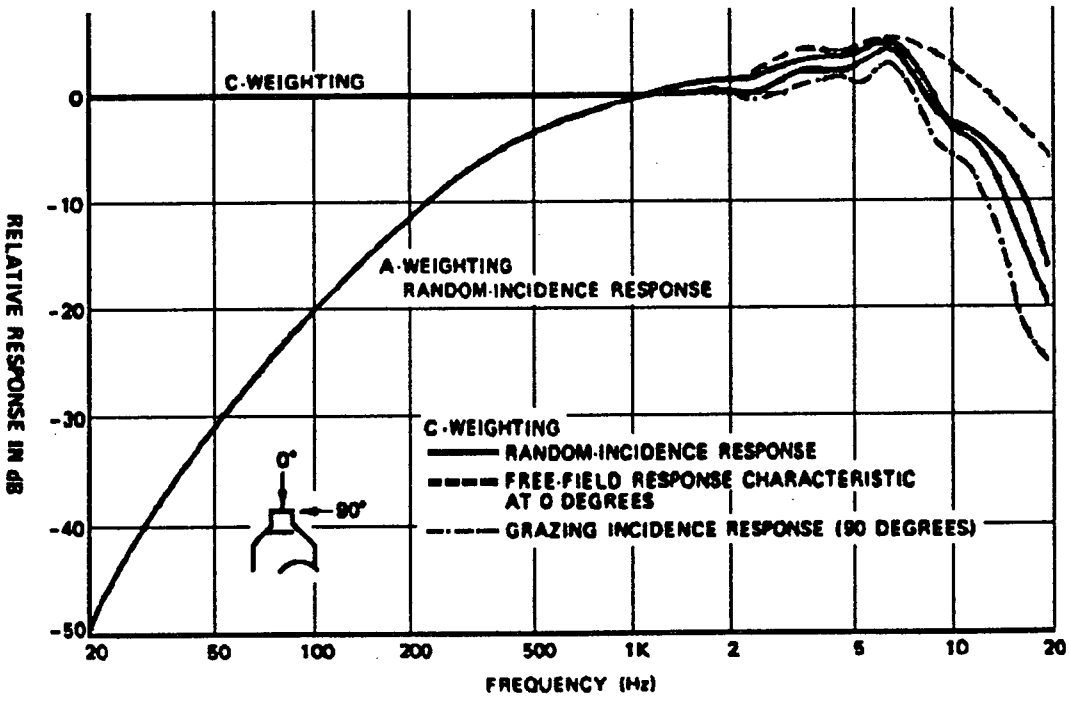
interrupt routine and the burst timing and spectral analysis are performed in real time as the input signal is digitized, the delay from the end of a burst, or interval between bursts, to the time a message is issued to the user is less than one 512 sample interval. This means that there is no appreciable time delay between the end of a warning signal's occurrence and the setting of the recognition flag.

6.2.2.1 Performance Measurements

Tests of the DSP implementation of the WARNSIS II algorithms were performed under quiet office conditions with an ambient noise level below 50 dB. The background noise added for testing was generated by a standard stereo receiver playing a combination of FM rock music and voices. Figure 6.5 (a) shows the physical location of signal and noise sources in relation to the microphone input. N shows the location of the noise generating source, W_A the location of the warning signal source A, W_B the location of the warning signal source B, M the microphone location and S the location of the sound level meter. Warning signal source A was a typical telephone ring generated by a Panasonic PD-2300 telephone whose FFT spectrum and characteristic peak frequency are shown in Figure 6.6. Warning signal source B was a continuous 2200 Hz tone generated by a function generator whose FFT spectrum and characteristic peak frequency are shown in Figure 6.7. Sound level measurements were made with a Realistic (TM) Sound Level Meter with both standard A and C weighting. As shown in Figure 6.5 (b) the C weighting curve has a flat frequency response from 32 Hz to approximately 2,500 Hz, the A weighting curve has a 3 dB highpass roll-off at approximately 500 Hz. This means that more low frequency energy is measured with the C weighting than for the A weighting.



(a) Noise, Warning Signal and Microphone Test Setup



(b) A and C Sound Level Meter Weightings

Figure 6.5 - Performance Measurement Setup

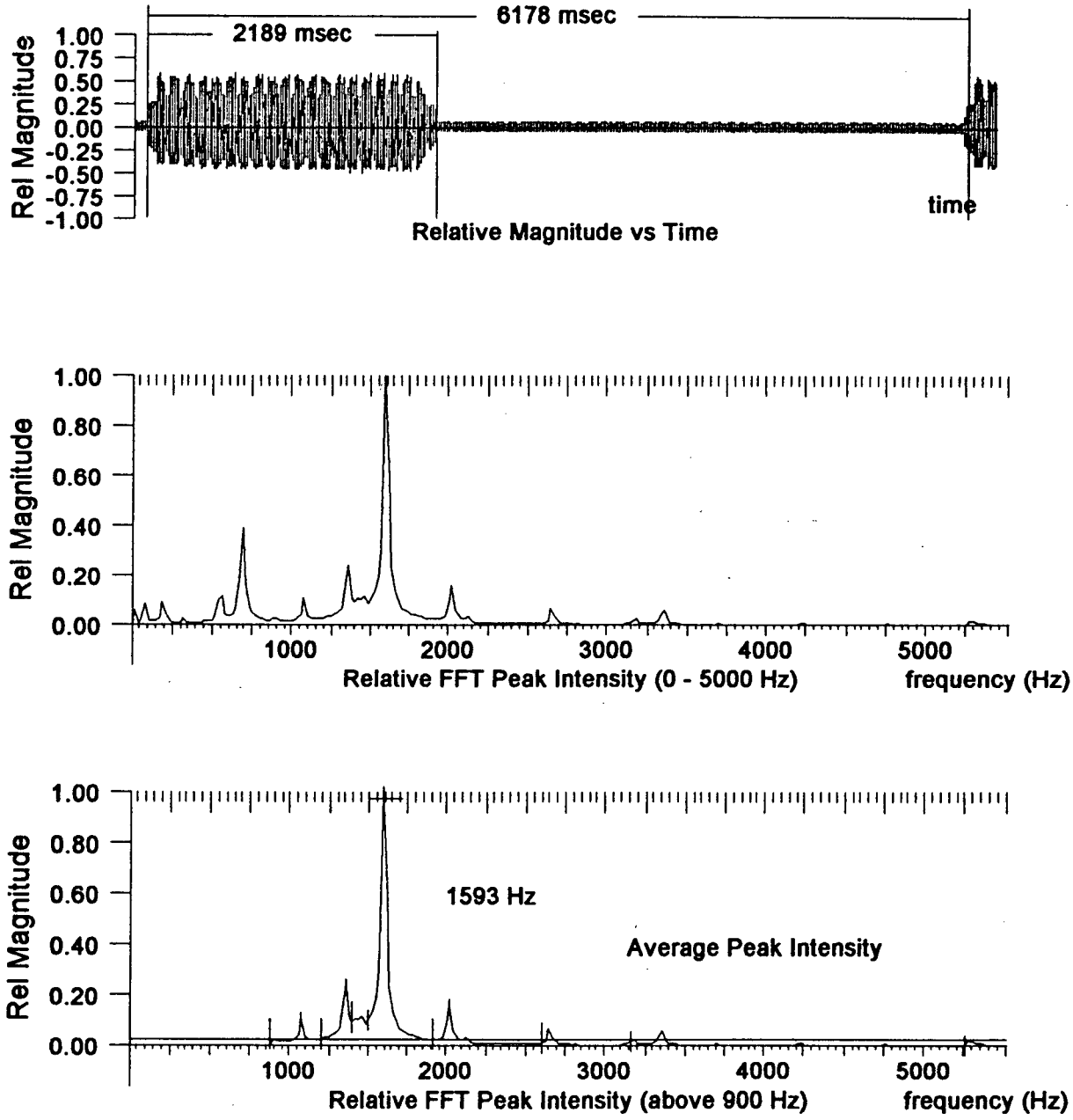


Figure 6.6 - Panasonic PD-2300 Electronic Ringing Signal Spectral Analysis

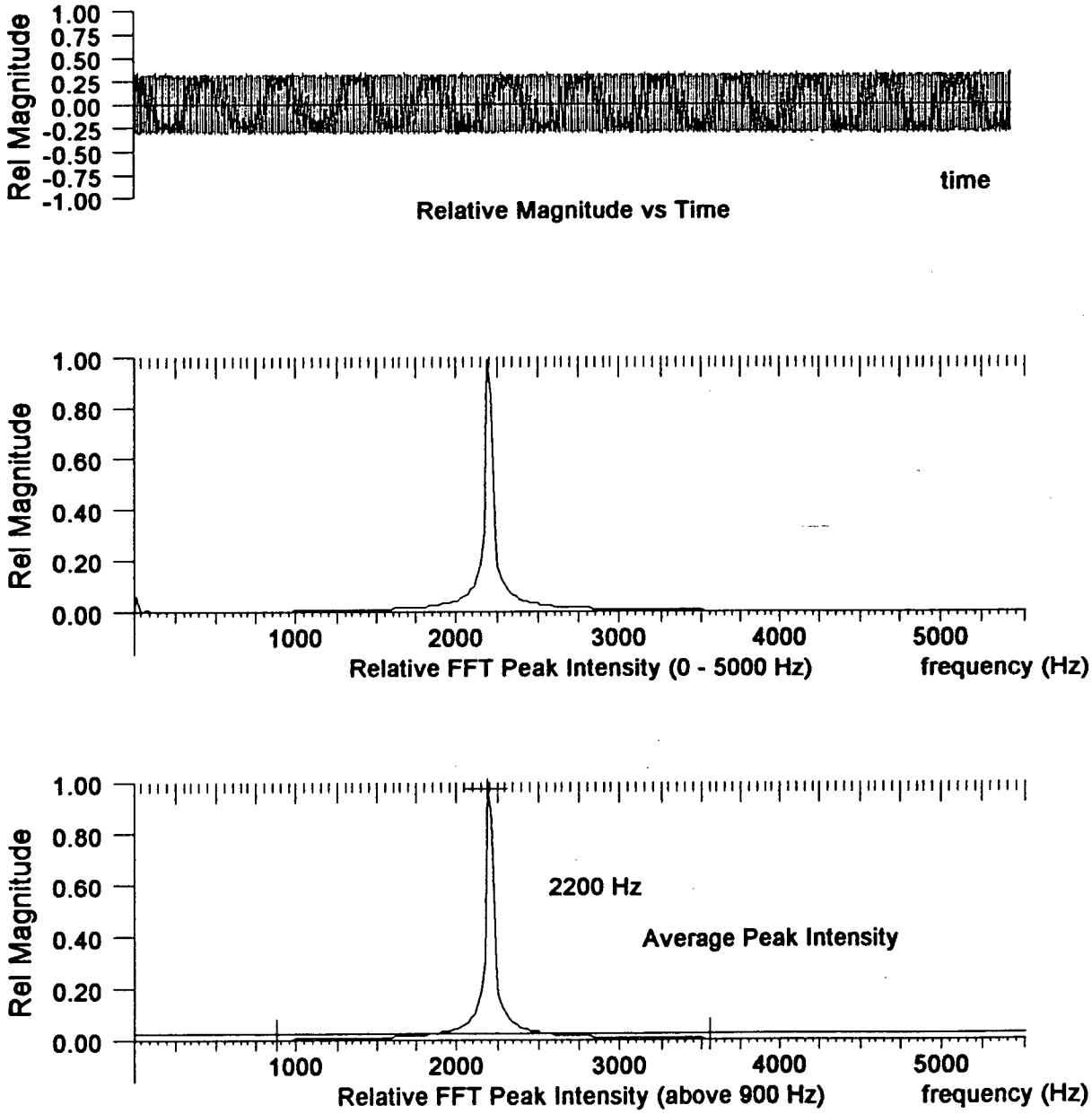


Figure 6.7 - 2200 Hz Continuous Tone Spectral analysis

Table 6.1 shows the upper limits on signal and noise for the DSP implementation of WARNSIS II for sound level measurements with both A and C weightings. As can be seen, the DSP implementation of the WARNSIS II algorithms can successfully detect warning signals in *very* noisy environments. The DSP implementation was able to detect a 57 dB warning signal at a distance of 4 ft from the device generating the warning signal in the presence of 76 dB average background noise. This is a signal-to-noise ratio of -19 dB. Since most warning signals are designed to be at least 10 dB above ambient noise levels, the DSP implementation far exceeds the requirements. In fact, in testing, the DSP implementation of WARNSIS II outperformed the author's own ability to detect warning signals by ear and to discriminate between warning signals in very noisy environments. The DSP implementation of the WARNSIS II algorithms performed with 100% recognition accuracy during testing. All occurrences of perviously learned warning signals were recognized until the background noise level exceeded the values shown in Table 6.2.2.1 below and no false positives were generated.

Table 6.1 - Single Warning Signal vs. Noise Sound Level Measurements

Distance a (ft)	Distance b (ft)	Meter Weighting	Warning Signal Strength (dBA)	Maximum Average Noise Level (dBA)	Maximum Peak Noise Level (dBA)
11	2	A	56	72	74
11	2	C	58	80	84
11	4	A	56	62	66
11	4	C	57	76	78

To determine if the presence of one warning signal would suppress the detection of another signal in a noisy environment, a second signal was introduced during testing. In order to determine the level at which the second warning signal (B) would suppress the detection of the original warning signal (A) the following procedure was used. First the noise background level was increased to the point where detection of warning signal A was suppressed by the noise alone. Next the noise level was decreased by 2 dBA to allow detection of warning signal A. The level of warning signal B was then increased to the point where detection of warning signal A was again suppressed. Table 6.2 shows the upper limits for the sound level for the two signals and the background noise.

Table 6.2 - Dual Warning Signal Vs Noise Sound Level Measurements

Distance a (ft)	Distance b (ft)	Meter Weighting	Warning Signal A Strength (dBA)	Warning Signal B Strength (dBA)	Maximum Average Noise Level (dBA)	Maximum Peak Noise Level (dBA)
11	2	A	58	58	70	72
11	2	C	62	58	78	82
11	4	A	58	55	60	62
11	4	C	60	55	70	74
11	4	A	58	55	-	-
11	4	C	60	55	-	-

The first four rows of table 6.2 show the level at which the detection of warning signal A was suppressed by warning signal B in the presence of background noise. The last two rows of table 6.2 show the level at which detection of warning signal A was suppressed by warning signal B alone, without the addition of background noise. It is significant to note that the presence or absence of the background noise has no effect on level at which suppression of signal A by signal B occurs.

Figure 6.8 shows the FFT spectrum of the background noise. Note that most of the energy is in the range below 900 Hz. Figure 6.9 shows the combined spectrum of signal A and signal B at the point just before suppression of signal A occurs. This shows that suppression of signal A occurs when the magnitude of the characteristic peak for signal B is approximately four times that of signal A. The noise signal can only contribute to the suppression of signal A by signal B if there is significant energy in the noise spectrum at the characteristic frequency peak of signal B. This is in agreement with the original algorithm design as discussed in section 5.3.

It would appear from the sound level readings that the strength of both signal A and signal B are at approximately the same level, 55 dBA to 60 dBA, and the magnitude of their characteristic peaks should also be approximately the same. However, since signal B is a pure tone, all of the energy in signal B is found at its characteristic frequency. Signal A, on the other hand, has its energy distributed across the spectrum, including a portion below 900 Hz, which means less energy is contained at the characteristic frequency .

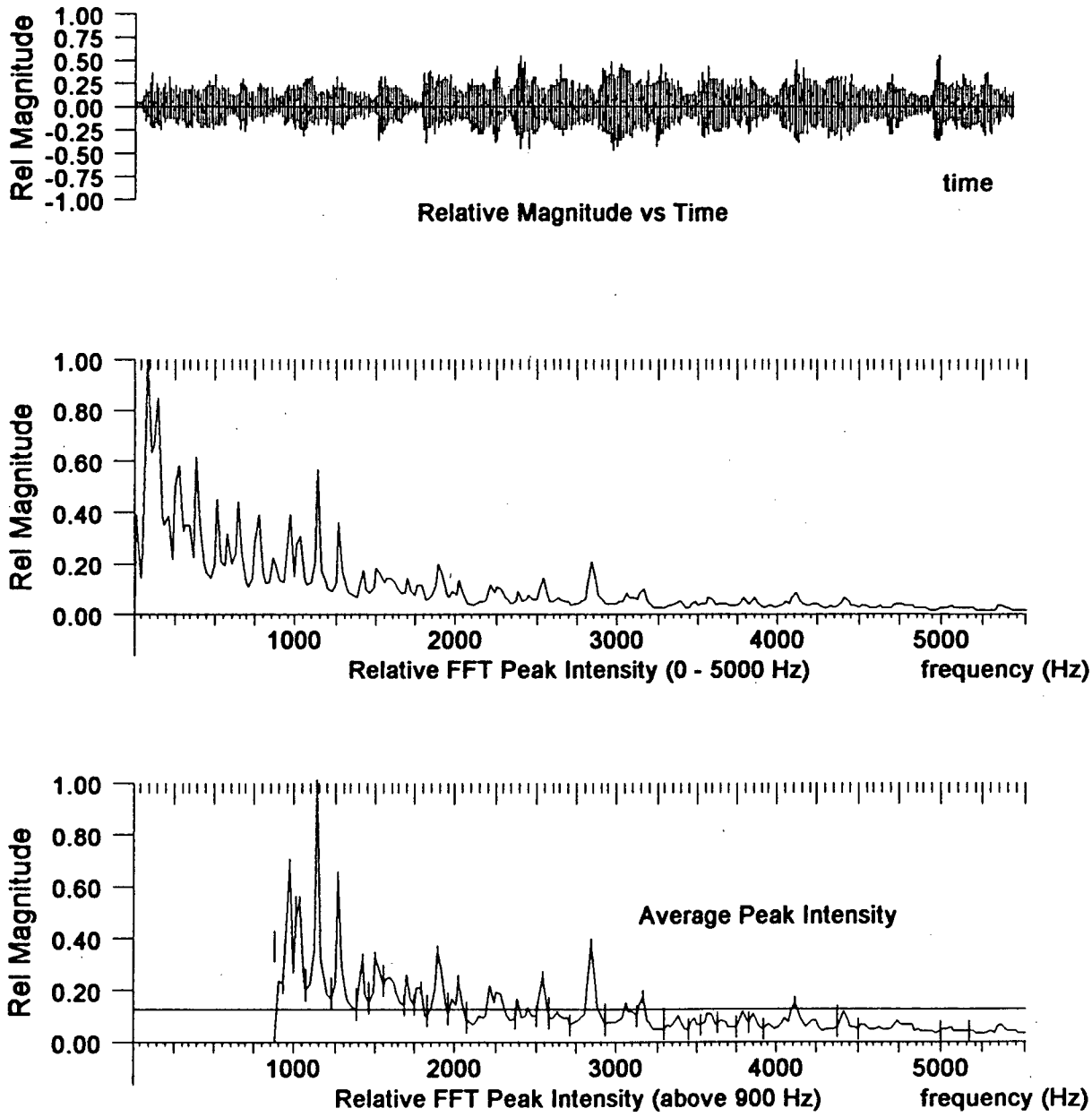


Figure 6.8 - Test Background Noise Spectral Analysis

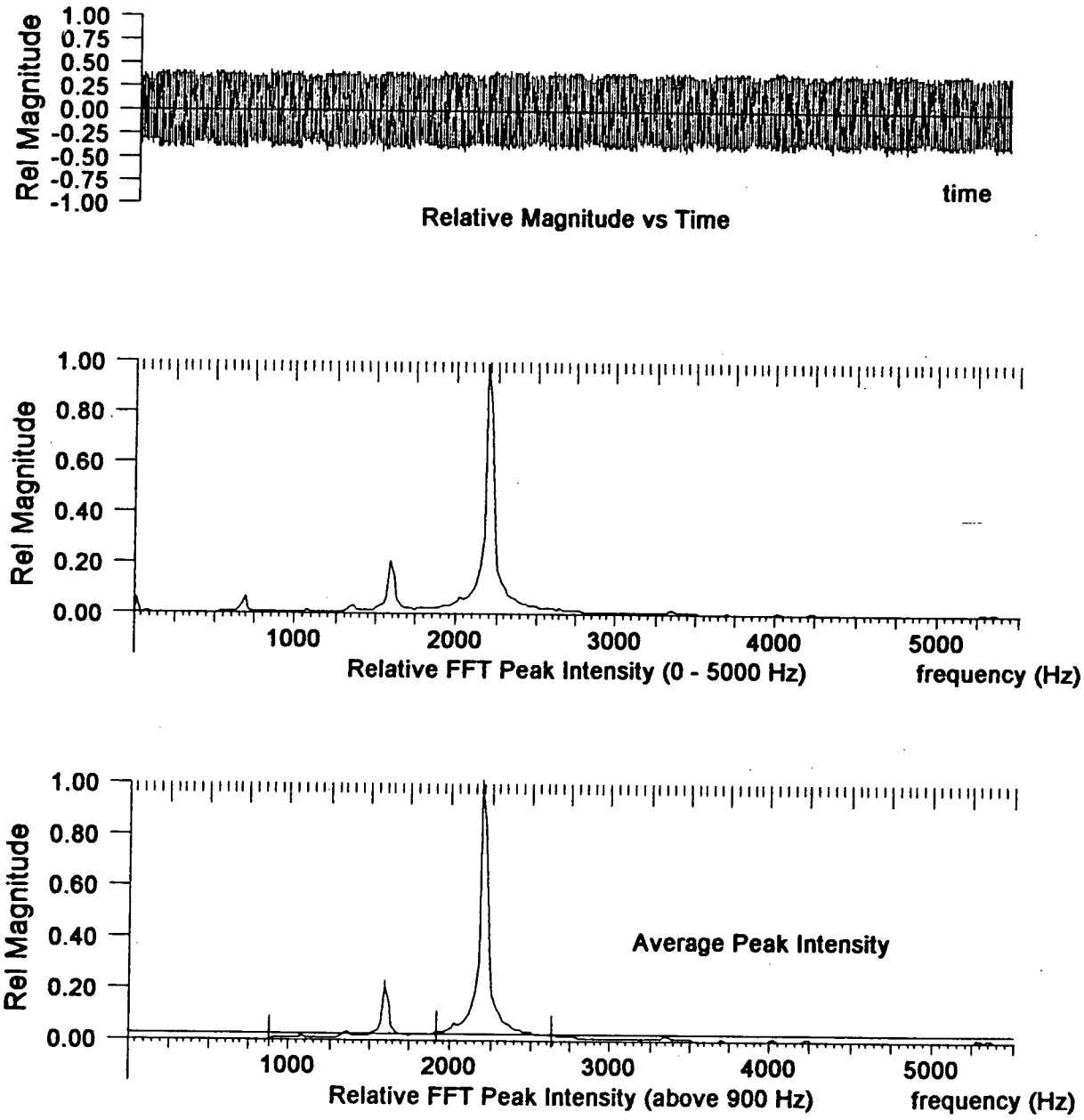


Figure 6.9 - Combined Warning Signal A and B Spectral Analysis

Chapter 7

7.0 Conclusions and Recommendations

It has been shown that the WARNSIS II algorithms, can, by the application of separate spectral, energy and timing analysis routines, in that order, learn and then later recognize warning sounds with 100% accuracy in very noisy background environments. The signal-to-noise ratio in these environments can be on the order of -20 dB. It has also been shown that warning signals can be successfully recognized on the basis of four characteristics:

i) peak frequency

ii) energy content at the peak frequency

iii) burst duration

iv) interval time.

Both continuous and burst warning signals can be recognized by this method. In particular, a burst signal can be recognized after a single burst and a repetitive warning signal can be successfully recognized on the basis of a single repetition. A warning signal can also be identified with no appreciable delay due to processing after the signal burst has ended. Additionally, it has been shown that it is possible to successfully recognize multiple overlapping warning signals in the presence of background noise. These results mean that it would be possible to implement a device that could operate successfully in real-life situations.

The implementations of the WARNSIS II algorithms on both a PC and a DSP module have also shown that it is possible to implement a system that does not rely on any specialized hardware and can be easily ported to distinctly different platforms.

In addition to aiding the design and development of the WARNSIS II algorithms, the PC implementation has provided, not only a tool for further development of WARNSIS II based systems, but a tool for general spectral analysis as well. The base routines and functionality of the PC program as well as the graphical displays for instantaneous signal amplitude, absolute signal amplitude and FFT power spectrum can easily be used as the starting point for other projects in this field.

The DSP implementation further shows that it is possible to implement a low cost, low part-count system, with sufficient power to perform the required analysis. The EVM board used in the DSP implementation contains only 24 IC's and provides extra functionality such as analog output and secondary serial communications not required for the WARNSIS II implementation. Although the EVM provided 16K of additional zero wait state RAM, less than 2K of this RAM was required for the WARNSIS II algorithms. This means that a stand-alone implementation of WARNSIS II could be produced on a system consisting of fewer than the 24 IC's on the EVM. Since the code and data space and processing time requirements for the WARNSIS II algorithms are small, it may be possible to add the WARNSIS II functionality to existing equipment without additional memory or processing power.

7.1 Future Work

Although the WARNSIS II system performed extremely well and met all of the objectives of phase II of the project, there are some areas where further improvement is possible.

7.1.1 Improved Learning Mode

Although the WARNSIS II recognition mode is able to operate in extremely high noise background areas, the learning mode algorithms require that the warning signal that is being learned be significantly louder than the background noise. In most cases this would not be a problem as it should be possible to arrange, temporarily at least, for a sufficiently quiet background in which to learn a warning signal. There may be cases, however, where this would not be possible and the warning signal would have to be learned in a high noise background environment.

It may be possible to improve the learning mode's ability to learn new signals in high noise environments by using an approach similar to the recognition mode's warning signal "candidacy" algorithm. Rather than relying on a single burst and using the STAAA to identify the beginning and end of a burst, multiple signal bursts and frequency characteristics could be used. For example, the first burst could be used to identify the characteristic frequency for the warning signal by monitoring for the occurrence of a strong peak in the power spectrum. The second burst could then be used for confirmation of the spectral peak and burst timing and a third complete burst could be used for interval timing and confirmation of the characteristic frequency and burst times.

If the WARNSIS II device were provided with enough memory to store the digitized input signal for two complete bursts, the learning mode could be designed to learn from a single repetition of a warning signal by performing the spectral analysis twice on the initial burst. The

first pass would identify the characteristic peak, the second pass would be used to identify the burst duration time. At least one partial repetition would still be required, however, to identify the interval time.

7.1.2 Improved Recognition Mode

It was stated earlier in section 5.3, and shown in section 6.2.2.1, that the presence of a warning signal that was significantly greater in power than other warning signals could, in effect, suppress the recognition of those other warning signals should they occur at the same time. It may be possible to overcome the suppression by performing a multi-pass frequency analysis. In the first pass, the peak with the largest magnitude in the spectrum would be used for normalizing the spectral magnitudes. Once the spectrum had been analysed using this peak for normalization, the process would be repeated using the next largest peak to normalize the spectrum. This would continue until all peaks that exceed the preset minimum "background" value had been used for normalization. This way a single powerful signal would not be able to suppress a valid peak. This would, however, require an increase in both processing power and memory. The processing would only require extra time to recalculate the normalized magnitudes for each pass, not to recalculate the FFT. The data storage requirements would increase by 8 words (256 bits, one for each frequency *bin*) for flags to indicate if a peak had been previously used for normalization.

7.1.3 Additional Features

We have shown that a WARNSIS II device is capable of correctly identifying warning sounds in noisy environments. There is often other information conveyed with a warning signal

in addition to that simple presence or absence of the sound. Humans rely on binaural hearing for sound localization. An improved WARNSIS II device could supply even more information to the user if some type of sound localization could be performed. Additionally, an estimate of the "loudness" of the warning signal could provide added benefit to the user. The implementation of these features should be explored.

7.2 Other Applications

As stated earlier the functionality of the WARNSIS II algorithms can be incorporated into other devices. This could provide assistance not only to the hard of hearing or the deaf but also to people in other situations. The problem of recognition of warning sounds is not limited to those with hearing impairments. There are situations which arise on a day to day basis which can have serious affect on people with normal hearing. In many industrial environments auditory warning signals are commonly used. On construction or mining sites it is common practice to use auditory warning signals to signal blasting or movement of heavy machinery. In mills and chemical processing plants the starting of a piece of equipment is often preceded by an auditory warning burst. Often the noise present in these environments and the need for personnel to wear hearing protection makes it difficult to hear these signals. Personnel who are working in an area where the risk of injury is high would benefit from a device that was able to alert them to these signals.

Another possible use for the WARNSIS II system is in hospitals. In critical care areas, patients may be connected to a number of different medical devices, each with its own alarm sound. Rather than relying on separate recording instruments or hard-wiring the alarms to the central station, a WARNSIS II device which monitors all alarms and reports to the central

station could save both wiring and training costs. Time and effort would also be saved when new equipment was introduced to a critical care area. Since each medical device would have its own warning signal characteristics which could be "pre-learned" by the WARNSIS II device, no extra connections or training would be required each time the equipment was introduced into a new area. By elimination of the hard-wire requirements for some devices, the space near the patient would be less cluttered, allowing for better patient access.

References

[1] "The Prevalence of Hearing Impairment and Reported Hearing Disability Among Adults in Great Britain", A.C.Davis, pp. 911-917, International Journal of Epidemiology, 18 (4)

[2] "Canadians with Impaired Hearing", J. Shein, ed., Publication Division, Statistics Canada, 1992

[3] "Visual Devices For Deaf and Hard of Hearing People: State-of-the-Art", Judith Harkins, GRI Monograph Series A, No. 3, Gallaudet Research Institute, 1991

[4] "Technological Devices for Deaf-Blind Children: Needs and Potential Impact", A.Y.J. Szeto, K.M.Christensen, IEEE Engineering In Medicine and Biology, Volume 7, pp. 25-28, September 1988

[5] "A Computer Sound Cue Indicator for Hearing Impaired People", C.Jensema, D.Hinton, N.Drew, pp. 49-52, Computing Applications to Assist Persons with Disabilities, 1992 Proceedings

[6] "Sound Alerting Aids for the Profoundly Deaf", B.Uvacek, G.S.Moschytz, pp. 2615-2618, IEEE International Symposium on Circuits and Systems, 1988

[7] "A Warning Signal Identification System (WARNSIS) for the Hard of Hearing and the Deaf", Simon Chau, Masters Thesis, University of British Columbia, 1989

[8] Speech Level and Spectrum, Electronics Engineer's Handbook, Second Edition, Section 19.10, pp.19-7, McGraw-Hill, 1984

[9] "Standard for the Installation, Maintenance and Use of Local Protective Signalling Systems for Guard's Tour, Fire Alarms and Supervisory Service", National Fire Protection Association, NFPA 72A, 1985

[10] "Guide for the Installation, Maintenance and Use of Notification Appliances for Protective Signalling Systems", National Fire Protection Association, NFPA 72G, 1985

[11] "Long-Duration Signal Detection in a Noisy Environment", S.C.Clontz, R.R.Adhami, pp. 527 - 532, IEEE 1989 South Eastern Symposium on System Theory, 1989

[12] "Speech Recognition Techniques", P.M.Grant, pp. 37 - 48, Electronics & Communications Engineering Journal, February 1991

[13] "Algorithms and Architectures for Dynamic Programming on Markov Chains", W.G.Bliss, L.L.Scharf, pp. 900-912, IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 37, No. 6, June 1989

[14] "A Real Time Connected Word Recognition System", Y.Ishikawa, K.Nakajima, pp. 215-217, 10th International Conference on Pattern Recognition, 1990

[15] "Speech Technology At Work", J. Hollingum & G. Cassford, IFS Publications, 1988

[16] "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", L.R.Rabiner, Proceedings of the IEEE, pp. 257-285, Vol 77, No. 2, February 1989

[17] "An Implementation of FFT, DCT, and Other Transforms on the TMS320C30", P. Papamichalis, pp. 53-136, Digital Signal Processing Applications with the TMS320 Family, Theory, Algorithms and Implementations, Texas Instruments, Volume 3, 1990

[18] "How the FFT Gained Acceptance" , J.W. Cooley, IEEE Signal Processing Magazine, Volume 9, Number 1, 1992

[19] "Numerical Recipes in Pascal", W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Cambridge University Press, 1992

[20] "A Spectral Magnitude Analysis Theorem and Applications", J.C.Anderson, pp 277-280, Time-Frequency and Time-Scale Analysis, IEEE International Symposium on Signal Processing, 1992

[21] "An Algorithm for the Machine Calculation of Complex Fourier Series", J.W. Cooley & J.W. Tukey, Math. Comp. 19, 1965

[22] "A New Look at the Comparison of the Fast Hartley and Fourier Transforms", M.Popović, D.Šević , pp. 2178 - 2182, IEEE Transactions on Signal Processing, Vol. 42, No. 8, August 1994

[23] "Transforming Real-Valued Sequences: Fast Fourier versus Fast Hartley Transform Algorithms", P.R.Uniyal, pp. 3249 - 3255, IEEE Transactions on Signal Processing, Vol. 42, No. 11, November 1994

[24] "Restructured Recursive DCT and DST Algorithms", P.Lee, F.Huang, pp. 1600 - 1609, IEEE Transactions on Signal Processing, Vol. 42. No. 7, July 1994

[25] "An Adaptive IIR Structure for the Separation, Enhancement, and Tracking of Multiple Sinusoids", C.C.Ko, C.P.Li, pp. 2832 - 2834, IEEE Transactions on Signal Processing, Vol. 42, No. 10, October 1994.

[26] "Linear and Quadratic Time-Frequency Signal Representations", F. Hlawatsch, G.F. Boudreaux-Bartels, IEEE Signal Processing Magazine, Volume 9, Number 2, 1992

[27] "A Wavelet Magnitude Analysis Theorem", J.C.Anderson, pp. 3541 - 3543, IEEE Transactions on Signal Processing, Vol. 41, No. 12, December 1993

[28] "Wavelets and Signal Processing", O.Rioul, M.Vetterli, pp. 14 -35, IEEE Signal Processing Magazine, October 1991

[29] "TMS320C3x Evaluation Module", Technical Reference, Texas Instruments, 1990

[30] "Performance and Compatibility Requirements for Telephone Sets", Canadian Standards Association, National Standard of Canada, CAN/CSA-T510-M87, March 1987.

[31] TLC32044C, TLC32044I Voice-Band Analog Interface Circuits, Linear Circuits Data Conversion, DSP Analog Interface, and Video Interface Data Book, Volume 2, Texas Instruments, 1992

[32] "Advanced Digital Signal Processing", J.G. Proakis, C.M. Rader, F. Ling, C.L. Nikias, pp. 498 - 520, MacMillan Pubilishing Company, 1992

Appendices

Appendix - A WARNSIS II TMS320C30 Assembly Language Code

```
*************************************************************************
* FILENAME          : warnsis.asm
*
* WRITTEN BY        : Kim Dotto
*
* DATE              : 23rd June 1994
*
* VERSION           : 1.0
*
*************************************************************************
* VER   DATE             COMMENTS
* --- ------------ ------------------------------------------------
* 1.0   23rd June 94     TMS320C30 Implementation of WARNSIS II project
*
*
*
*
*
*************************************************************************
*
* SYNOPSIS:
*
*
* The WARNSIS Program is designed to detect the presence of audible warning
* signals.  These signals can be single or repetative burst type in nature,
* or they can be continuous. Typical devices generating these warning
* signals would be telephones, microwaves, smoke detectors or other home
* or office devices.
*
* The WARNSIS II algorithms eliminate the need for specialized hardware and
* provides very good performance in noisy environments. Spectral analysis
* is performed continuously by software using the FFT instead of specialized
* chipsets.  The start or end of a warning signal burst or repetition is
* determined by the presence or absence of key spectral information for
* a given warning signal.  The FFT power spectrum for the realtime input
* is compared to information stored in predefined templates.  If the

* spectral information matches the template a spectral match is registered.
* The signal is then continuously monitored until the spectral signature
* disappears.  The burst duration and interval times are then compared to
* those in the template.  If a match occurs for both spectral and burst
* duration then a burst match is indicated.  If a match also occurs for
* the interval time then an interval match is indicated.
*
* Because the spectral analysis is carried out in software, the WARNSIS II
* project can be ported to any hardware with sufficient power to perform
* the FFT analysis in realtime.  This includes a 50MHz 486 PC running under
* MS Windows, where the software was originally designed.  The Windows
* program that performs the analysis is called WARNSIS.EXE and was written
* for Borland's Turbo PASCAL for Windows Ver. 1.5.  This program and the
* accompanying source files should be available from the same source as
* this file.
```

```
*
* This version of the WARNSIS II project is desgined to run on the Texas
* Instruments TMS320C30 DSP and the C30 Evaluation Module and is intended
* to show that the project can be implemented on a low cost DSP chip, not
* just an Intel 80C486 50MHz system.
*
*
*************************************************************************
*
* FILES:
*
*   WARNSIS.ASM          Main program file
*   WARNMATH.ASM         Math routines for Floating Point Divide
*   WARNFFT.ASM          FFT subroutine code file
*   WARNTWID.ASM         FFT twiddle factors file (sine_table)
*
*   WARNFFT.CMD          Linker command file
*
*************************************************************************
*
*  PASCAL PSEUDOCODE:
*
* program warnsis;
*
* const
*
*  sine_table = array [0..255] of real; {twiddle factors}
*  min_hz     = 1000;                    {minimum cutoff frequency}
*  max_hz     = 4000;                    {maximum cutoff frequency}
*  min_power  = 1000;                   {minimum spectral peak power below}
*                                        {which peak will not be considered}
*
* var
*  i,j      : integer;                  {general purpose counters}
*  fill_cnt : integer; {current position of table being filled}
*  fftdat0,fftdat1 : array[0..511] of real; {2 tables for collecting input}
*                                           {and calculating fft power   }
*                                           {spectrum info               }
*  ffttotal : array [0..511] of real; {storage for total fft spectrum power}
*                                     {used in learning mode               }
*  max_fft  : real; {maximum fft magnitude in power spectrum,used for   }
*                   {normalization of power spectrum between samples    }
*  ave_fft  : real {average fft magnitude in power spectrum, gives      |
*                  {baseline for comparison of peak to background noise |
*
*  {Each new type of signal must be learned before it can be detected }
*  {each "learned" signal has its own template with the following info}
*
*   template = record
*    info : string[30];              {template name}
*    burst_duration : real;          {burst duration}
*    interval_time  : real;          {interval duration}
*
*    waiting : boolean;              {currently matching spectrum flag}
*   end;
*
*
*  procedure ffft_rl(fft_size, log_size: integer; source_addr, dest_addr,
```

```
*                sine_table : ptr;  bit_reverse: boolean );
*  begin
*   if source_addr=dest_addr then
*    calculate_fft_inplace
*   else
*    calculate_fft;
*   if bit_reverse then do_bit_reversals;
*  end;
*
*  {get_realtime_data id called upon end of conversion interrupt}
*
*  procedure get_realtime_data : interrupt
*   begin
*    tmp_real:=get_analog_value;
*    if storing_table0 then
*     begin
*      fftdat0[fill_cnt]:=tmp_real
*     end
*    else
*     begin
*      fftdat1[fill_cnt]:=tmp_real
*     end;
*    inc(fill_cnt);
*    start_next_conversion;
*   end;
*
*  begin {main program}
*   initialize_data_structures;
*   start_first_A/D_conversion;
*   repeat
*    if learning_mode then
*     begin
*      cancel_template_tests;
*      if check_for_available_template=true then
*       begin
*        initialize_total_fft_power_array;
*        wait_for_input_signal_above_predefined_amplitude;
*        start_duration_timer;
*        start_interval_timer;
*        while input_above_limit do
*         begin
*          calculate_fft_every_512_samples;
*          add_current_fft_magnitudes_to_total_fft_magnitudes
*         end;
*        end_duration_timer;
*        save_duration_time;
*        find_main_peak_in_fft_power_spectrum;
*        save_peak_frequency_and_peak_width;
*        wait_for_input_signal_above_predefined_amplitude_again;
*        if second_amplitude_increase_found then
*         begin
*          end_interval_timer;
*          save_interval_time;
*         end
*        else
*         cancel_interval_timer;
*        exit_learning_mode;
*       end
```

```
*       else
*        issue_warning;
*      end
*     else
*      begin
*       if fill_cnt=512 then
*        begin
*         fill_cnt:=0;
*         set_other_table_to_be_filled;
*         if fftdat0_full then
*          ffft_rl(512,9,fftdat0,fftdat0,sine_table,false)
*         else
*          ffft_rl(512,9,fftdat1,fftdat1,sine_table,false)
*
*         {|fft| and find the maximum and average value of table for those}
*         {fft values for frequencies between max_hz and min_hz }
*         ave_fft:=0
*         max_fft:=0;
*         j:=0;
*         for i:=0 to 255 do
*          if fft_hz[i]>min_hz and fft_hz[i]<max_hz then
*           begin
*            inc(j);
*          Fft_real[i]:=
*                sqrt(fft_real[i]*fft_real[i]+fft_imag[i]*fft_imag[i]);
*            if fft_real[i]>max_fft then max_fft:=fft_real[i];
*            ave_fft:=ave_fft+fft_real[i];
*           end;
*         ave_fft:=ave_fft/j;
*
*         identify_peaks;{identify peaks in spectrum > 0.25*max_fft}
*         for all_templates do
*          begin
*           for each_peak_found do
*            if (abs(peak_freq-template_freq)<template_tolerance) and
*               (peak_magnitute>2.5*ave_fft) and
*               (peak_magnitude>min_power) then {found a spectral match}
*             begin
*              if currently_timing_a_spectral_match_for_this_template then
*               begin
*                if have_been_timing_for_at_least_2_sample_periods then
*                 set_flag_to_indicate_valid_start_of_signal;
*                continue_timing_spectral_match;
*               end;
*              else
*               begin
*                start_timing_spectral_match;
*               end
*             end
*            else
*             begin
*              if currently_timing_a_spectral_match_for_this_template then
*               if have_lost_the_match_for_at_least_2_sample_periods then
*                begin
*                 stop_timing_spectral_match;
*                 if timing_duration=template_value+-tolerance then
*                  issue_duration_match_message;
*                 if currently_timing_interval then
```

```
*                 if interval_duration=template_value+-tolerance then
*                  issue_interval_match_message;
*                reset_interval_timer_for_this_template;
*               end;
*             end;
*          end;
*        end
*       else
*        wait_for_full_table
*      end;
*    until done;
*  end;
*
*
*************************************************************************

        .global sinetab
        .global process_mode,fill_cnt,r1a_addr,r1b_addr
        .global _ffft_r1
        .global SQRT
        .global FINV
        .global FDIV
        .global find_peaks,fft_magnitude,find_match,fill_array,get_data
        .global start_learning,start_analysis,idle_loop,setup_fft
        .global fcnt,wfill,process_mode
        .global max_fft,ave_fft,cycle_cnt
        .global r1a_addr,r1b_addr,fft_addr
        .global t_info,t_config,t_burst_dur,t_interval_time
        .global t_range_min,t_range_max,t_range_tol,t_range_peak
        .global t_waiting
        .global peak_hz,peak_cnt,peaks_hcnt,peaks_tcnt,g_up,peak_max
        .global peak_cs
        .global t2000
        .global t_sig
        .global shortmix
        .global microsht
        .global no_message
        .global test_burst,test_interval,template_loop
        .global case1_0,case1_1,case1_2,case1_3,case1_4,case1_end
        .global case2_0,case2_1,case2_2,case2_3,case2_4,case2_end

        .global case3_0,case3_1,case3_2,case3_3,case3_4,case3_5,case3_end
        .global case4_0,case4_1,case4_2,case4_3,case4_4,case4_5,case4_end
        .global found_burst,found_int,data_index,max_peak
        .global ave_ampl,learn_cnt,keep_waiting,tmp_table
        .global no_fft_req,find_template,less_than_min,find_main_peak
        .global find_amplitude,sum_ffts,copy_fft_sum,pick_main_peak
        .global save_template_info,clear_table,average_fft_sum

        .global sysinit,aicreset,com_parm,hcontrol,receive0
        .global hread16,hwrite16,hread32,hwrite32
        .global wait_transmit_0, dmadone, wordflag

;Template Information for 5 templates

t_info          .usect  ".fftdata",5    ;template name
t_burst_dur     .usect  ".fftdata",5    ;burst duration
t_interval_time .usect  ".fftdata",5    ;interval duration
```

```
t_range_peak    .usect  ".fftdata",5   ;peak frequency
t_waiting       .usect  ".fftdata",5   ;currently matching spectrum flag
t_bstart        .usect  ".fftdata",5   ;burst timer start time
t_bend          .usect  ".fftdata",5   ;burst timer end time
t_int_count     .usect  ".fftdata",5   ;interval timer


process_mode    .usect  ".fftdata",1   ;analyze=0 learn=1
fill_cnt        .usect  ".fftdata",1   ;realtime data count flag
fcnt            .usect  ".fftdata",1   ;realtime data count
wfill           .usect  ".fftdata",1   ;realtime data array flag
cycle_cnt       .usect  ".fftdata",1   ;timing cycle counter
none_waiting    .usect  ".fftdata",1   ;no templates waiting flag

learn_cnt       .usect  ".fftdata",1   ;learning mode cycle counter
ave_ampl        .usect  ".fftdata",1   ;average input signal strength
test_ampl       .usect  ".fftdata",1   ;average input signal strength

peak_cnt        .usect  ".fftdata",1   ;number of peaks in spectrum
g_up            .usect  ".fftdata",1   ;going up a peak or down?
peak_hz         .usect  ".fftdata",60  ;frequency bin# of identified peak
peak_max        .usect  ".fftdata",60  ;power of identified peaks


r1a_addr        .usect  ".fftdata",1   ;table 0 address
r1b_addr        .usect  ".fftdata",1   ;table 1 address
fft_addr        .usect  ".fftdata",1   ;fft table address
data_index      .usect  ".fftdata",1   ;address index of real data

max_fft         .usect  ".fftdata",1   ;maximum value for normalization
ave_fft         .usect  ".fftdata",1   ;average fft magnitude

tmp_table       .usect  ".tmpdata",256 ;temporary table for summing ftts
max_peak        .usect  ".fftdata",1   ;peak with greatest magnitude in FFT
peak_cs         .usect  ".fftdata",10  ;frequency bin# of identified peaks
peaks_hcnt      .usect  ".fftdata",1   ;peak with greatest magnitude in FFT
peaks_tcnt      .usect  ".fftdata",1   ;peak with greatest magnitude in FFT
cont_flag       .usect  ".fftdata",1   ;continous signal flag


*************************************************************************


stack_size .set    500

stack    .usect  ".stack",stack_size

         .sect   ".vecs"
PARMS:
reset    .word   warnsis
int0     .word   cmd_write
int1     .word   null_int
int2     .word   null_int ;hread16
int3     .word   null_int
xint0    .word   transmit0
;rint0          .word   receive0
```

```
rint0    .word   fill_array

xint1    .word   transmit1
rint1    .word   recieve1
tint0    .word   null_int
tint1    .word   timer1
dint0    .word   null_int

************************************************************************

                .data

min_ampl        .float  500.0              ;minimum amplitude (500)
min_power       .float  3.0e6 ;6           ;minimum spectral peak power

hostport        .word   000804000h

ram0_addr       .word   809800H
ram1_addr       .word   809C00H
ram1a_addr      .word   809C00H
ram1b_addr      .word   809E00H
tmp_addr        .word   tmp_table         ;address temp calculation table
peak_table      .word   peak_hz           ;address of peak table
peak_mtable     .word   peak_max          ;address of peak maximum values
sine            .word   sinetab           ;address of twiddle factors
templates       .word   t_info            ;address of template info
t_rp            .word   t_range_peak      ;address of template peaks
t_bdur         .word   t_burst_dur      ;address of template burst durations
t_wait         .word   t_waiting        ;address of template waiting flags
t_start        .word   t_bstart         ;address of template burst start
t_end          .word   t_bend           ;address of template burst end
t_int          .word   t_interval_time   ;address of template interval time
t_intc         .word   t_int_count      ;address of template interval count
temp_def        .word   template_defaults ;address of template defaults
peak_ctable     .word   peak_cs           ;address of peak table
peaks_hcount    .word   peaks_hcnt        ;address of peak table index
peaks_tcount    .word   peaks_tcnt        ;address of peak table index

fft_voc_data    .word   microsht          ;address of realtime data

template_defaults
                .word   0          ;template names
                .word   1
                .word   2
                .word   3
                .word   4

                .word   13         ;burst durations
                .word   34
                .word   1000
                .word   1000
                .word   1000

                .word   26         ;interval times
                .word   92
                .word   1000
                .word   1000
                .word   1000
```

```
                .word   92          ;range peak
                .word   101
                .word   255
                .word   255
                .word   255

                .word   0           ;waiting state flags
                .word   0
                .word   0
                .word   0
                .word   0

                .word   0           ;burst timer start times
                .word   0
                .word   0
                .word   0
                .word   0

                .word   0           ;burst timer end times
                .word   0
                .word   0
                .word   0
                .word   0

                .word   0           ;interval timer start times
                .word   0
                .word   0
                .word   0
                .word   0

                .sect       ".aicdata"

;stack_addr .word   stack            ;address of stack
;int1_hwr16 .word   hwrite16         ;Address of 16 bit host write function
;int1_hwr32 .word   hwrite32         ;Address of 32 bit host write function
;int1_cwr   .word   cmd_write        ;Address of command write function
;int2_hrd16 .word   hread16          ;Address of 16 bit host read function
;int2_hrd32 .word   hread32          ;Address of 32 bit host read function
;cmd_temp   .word   com_cmd               ;Temporary address of command
parameters
;com_parm   .word   com_stat         ;Address of command parameters
;wordflag   .word   0

*****************************************************************
* Addresses of various peripherals and memory control registers *
*****************************************************************

;dma_ctl    .word   000808000h            ;dma global control register
mcntlr0    .word   000808064h        ;i/o interface control reg. addr.
mcntlr1    .word   000808060h        ;parallel interf. cntl. reg. addr.
t0_ctladdr .word   000808020h        ;Timer 0
t1_ctladdr .word   000808030h        ;Timer 1
p0_addr    .word   000808040h        ;Serial port 0

*****************************************************************
* Control parameters to large to fit in immediate value      *
*****************************************************************
```

```
enbl_eint1 .word   000020400h        ;int1 enterrupt dma (host writes)
enbl_eint2 .word   000040400h        ;int2 interrupt dma (host writes)
enbl_sp0_r .word   000000020h        ;serial port 0 receive interrupt
intoff     .word   0fff0fbf1h        ;turn off int1,int2,int3,eint0,eint1
                                     ;eint2, eint3, dma
intclr     .word   0fffffff9h        ;clr out int0-2
t0_ctlinit .word   0C00002C1h        ;set timer as clk out, H1/2 period
                                     ;timer will run when cpu stops in
                                     ;emulation mode

p0_global  .word   00e970300h        ;serial port 0 global control register

;dma_wctl  .word   0C0000943h       ;dma write control
                                    ;com. reg. -> C30 mem.
                                    ;interrupt driven from host writes
;dma_rctl  .word   0C0000A13h       ;dma read control
                                    ;c30 mem -> com reg
                                    ;interrupt driven from host reads

*********************************************************************
* Host communications command structure                             *
*********************************************************************

;com_stat   .word   0000000000h          ;command status
;com_cmd    .word   0000000000h          ;command
;com_countl .word   0000000000h          ;transfer count low
;com_counth .word   0000000000h          ;transfer count high
;com_saddrl .word   0000000000h          ;source addr low
;com_saddrh .word   0000000000h          ;source addr high
;com_daddrl .word   0000000000h          ;destination address low
;com_daddrh .word   0000000000h          ;destination address high


*********************************************************************
* Various constants                                                 *
*********************************************************************
WAIT0          .set    0000h       ;memory control reg val, parallel bus
WAIT1          .set    0000h       ;memory control reg val, i/o bus
CACHE          .set    1800h       ;clear and enable cache
ENBL_GIE       .set    2000h       ;global interrupt enable
ENBL_INT0      .set    0001h       ;interrupt 0 enable
ENBL_INT1      .set    0002h       ;interrupt 1 enable
ENBL_INT2      .set    0004h       ;interrupt 2 enable
ENBL_INT3      .set    0008h       ;interrupt 3 enable
ENBL_XINT0     .set    0010h       ;serial port 0 transmit int. enable
ENBL_RINT0     .set    0020h       ;serial port 0 receive  int. enable
ENBL_XINT1     .set    0040h       ;serial port 1 transmit int. enable
ENBL_RINT1     .set    0080h       ;serial port 1 receive int. enable
ENBL_TINT0     .set    0100h       ;timer 0 interrupt enable
ENBL_TINT1     .set    0200h       ;timer 1 interrupt enable
ENBL_DINT      .set    0400h       ;dma interrupt enable (cpu)

BEGIN_CMD_SEND .set    1           ;Begin sending cmd parameters
END_CMD_SEND   .set    2           ;End sending cmd parameters
INIT_DONE      .set    5           ;Reset intialization complete

CMD_OK         .set    0           ;Received cmd parameters ok
CMD_ERROR      .set   -1           ;Error on receiving command parameters
```

```
CMD_FINISH      .set    0               ;Status, command is finished
CMD_LOAD        .set    1               ;Status, command is being loaded
CMD_ACTIVE      .set    2               ;Status, command is currently active

CMD_NOP         .set   10               ;Nop command
CMD_HOST_MR16   .set   11               ;C30 memory read, 16 bit mode
CMD_HOST_MW16   .set   12               ;C30 memory write, 16 bit mode
CMD_HOST_MR32   .set   13               ;C30 memory read, 32 bit mode
CMD_HOST_MW32   .set   14               ;C30 memory write, 32 bit mode
CMD_HOST_DMAR   .set   15               ;C30 memory read via dma, 16 bit mode
CMD_HOST_DMAW   .set   16               ;C30 memory write via dma,16 bit mode


                .text

warnsis:
                xor     ie,ie
                xor     if,if


                xor     R0,R0                   ;clr R0
                ldp     @ram1_addr,DP           ;set data page
                ldi     @fft_voc_data,AR0       ;get address of real data
                sti     AR0,@data_index         ;save it in data_index
                ldi     @ram1_addr,AR0          ;load address of RAM1a block
                ldi     @sine,AR2               ;load address of sine table
                ldi     1023,RC                 ;load repeat counter for 1024
                rptb    end_init                ;repeats
start_init:
;                ldf     *AR0,R1                 ;get real data
;                subf    128,R1                  ;subtract 128 offset
                ldi     0,R1
end_init:       sti     R1,*AR0++               ;save adjusted real data
                                                ;start first A/D conversion
                ldi     @ram1a_addr,AR0         ;load address of RAM1a block
                ldi     @ram1b_addr,AR1         ;load address of RAM1b block
                ldp     @process_mode,DP        ;set data page
                sti     AR0,@r1a_addr           ;save table address current
                sti     AR1,@r1b_addr           ;data page
                sti     R0,@process_mode        ;set for normal operation;
                sti     R0,@fcnt                ;set counter to zero
                sti     R0,@peaks_hcnt          ;set peak table head pointer=0
                sti     R0,@peaks_tcnt          ;set peak table tail pointer=0
                ldi     @hostport,AR4
                sti     R0,*AR4                 ;send 0 to pc
                not     R0,R1
                sti     R1,@wfill               ;set initial table
;                sti     R1,@process_mode        ;set for learning mode;
                sti     R0,@fill_cnt            ;set fill_cnt to 0
                sti     R0,@cycle_cnt           ;set timer count to 0
                sti     R0,@none_waiting        ;set no templates waiting to 0
                ldi     @templates,AR0          ;load address of template info
                   ldi      @temp_def,AR1           ;load address of template
defaults
                ldi     39,RC
                rptb    end_temp                ;repeats
                ldi     *AR1++,R1
```

```
end_temp:       sti     R1,*AR0++           ;set up default templates

                ldp     PARMS
                or      ENBL_GIE,ST         ;enable global interrupt
                or      ENBL_XINT0,IE       ;enable serial port 0 tx int
;                or      ENBL_INT2,IE        ;enable serial port 0 tx int
                or      ENBL_INT0,IE        ;enable serial port 0 tx int
;                or      ENBL_RINT0,IE       ;enable serial port 0 rx int
                call    aicreset            ;routine to reset aic
                ldp     @process_mode,DP    ;set data page

;The processor waits here while it fills the arrays with data
;when an array is full then processing begins

idle_loop:
                xor     R0,R0
                cmpi    @process_mode,R0    ;are we in learning mode?
                beq     do_analysis         ;if we are do learning routine
                callu   start_learning
do_analysis:
;                call    fill_array          ;get data
                ldi     @fill_cnt,R0        ;else have we 512 pieces of
data?
                cmpi    512,R0              ;if we do process then
                calleq  start_analysis      ;else wait for more data
                br      idle_loop
end__idle:

;Analysis Subroutines

setup_fft:
                ldi     1,R0                ;set flag for doing bit reversals
                push    R0                  ;put it on the stack
                ldi     @sine,R0            ;get address of sine table
                push    R0                  ;put it on the stack
                ldi     @r1a_addr,R0        ;get address of data table 0
                ldi     @wfill,R1           ;get which table flag wfill
                not     R1,R1               ;not(wfill) to recover flag
                cmpi    0,R1                ;are we using table 0?
                beq     fft_0
                ldi     @r1b_addr,R0        ;no then get table 1 address
fft_0:
                sti     R0,@fft_addr        ;save location of fft when done
                push    R0                  ;push correct table address on
                push    R0                  ;stack for both source and dest.
                ldi     9,R0                ;log(fftsize):=9
                push    R0                  ;put it on the stack
                ldi     512,R0              ;fftsize:=512
                push    R0                  ;put it on the stack


;                xor     ENBL_GIE,ST         ;disable global interrupt
                call    _ffft_rl            ;calculate fft
                or      ENBL_GIE,ST         ;enable global interrupt

                pop     R0                  ;get rid of parameters for
                pop     R0                  ; _ffft_rl
                pop     R0
```

```
                pop     R0
                pop     R0
                pop     R0
                retsu

;Subroutine to perform Main Analysis for signal detection
start_analysis:
                xor     R0,R0
                sti     R0,@fill_cnt
;                ldi     @hostport,AR4       ;
;                sti     R0,*AR4             ;   send it to pc
                ldp     @process_mode,DP    ;set data page
                ldi     @cycle_cnt,R0       ;get cycle_cnt
                addi    1,R0                ;inc(cycle_cnt)
                sti     R0,@cycle_cnt       ;save cycle_cnt

                xor     ENBL_GIE,ST         ;disable global interrupt
                call    setup_fft

;                ldp     @ram1_addr,DP       ;set data page
;                ldi     @ram1_addr,AR0      ;load address of RAM1a block
               call    fft_magnitude       ;calc magnitude, maximum and ave
;                ldp     @ram1_addr,DP       ;set data page
;                ldi     @ram1_addr,AR0      ;load address of RAM1a block
                call    find_peaks          ;find peaks in spectrum

                call    find_match          ;find any template matches

end_analysis:   retsu


;Learning Subroutine

start_learning:
                push    DP
                push    AR0             ;template interval time
                push    AR1             ;template burst duration
                push    AR2             ;burst timer end times
                push    AR3             ;burst timer start times
                push    AR4             ;waiting flags
                push    AR5             ;template peak values
                push    AR6             ;peak table magnitudes
                push    AR7             ;peak table peak values
                push    R0
                push    R1
                push    R2              ;
                push    R3              ;
                push    R4              ;found flag
                push    R5              ;waiting flag
                push    R7              ;
                push    IR0             ;peak tables index
                push    IR1             ;template tables index

                ldp     @peak_hz,DP     ;set data page
                ldi     @peak_table,AR7 ;get address of peak table
                ldi     @peak_mtable,AR6 ;get address of peak max table
                ldi     @t_rp,AR5       ;get address of template peaks
                ldi     @t_wait,AR4     ;get address of template flags
```

```
                ldi     @t_start,AR3      ;get address of burst timer start
                ldi     @t_end,AR2        ;get address of burst timer end
                ldi     @t_intc,AR0       ;get address of interval timer

                ldi     0,IR1             ;template_index:=0
                ldf     -1.0,R0           ;
                stf     R0,@test_ampl     ;set test amplitude flag

                ldi     0E000h,R7
                call    write_queue       ;send flag for learning mode

                ;insert code here to
                ;cancel all waiting templates from process mode
                ;when entering learning mode, can be done while looking
                ;for blank template because all unused templates are after
                ;the templates that are being used

find_template:  ldi     *+AR5(IR1),R0
                cmpi    255,R0            ;if template_peak = 255 then
                beq     found_template    ; found blank template
                addi    1,IR1             ;else
                br      find_template     ; inc(index) and try again
found_template:
                ldi     0E020h,R7
                addi    IR1,R7
                call    write_queue       ;send flag for learning mode

                ;save cycle_count here to set absolute time limit on
                ;learning cycle to recover from unknown signal

                ldi     @cycle_cnt,R0
                sti     R0,@learn_cnt

                xor     R0,R0
                sti     R0,@cont_flag      ;continuous signal flag
                sti     R0,*+AR2(IR1)     ;zero burst end
                sti     R0,*+AR3(IR1)     ;     burst start
                sti     R0,*+AR0(IR1)     ; and interval times
                call    clear_table       ;clear fft sum table

keep_waiting:
;                call    fill_array        ;get data
                ldi     @fill_cnt,R0      ;do we have 512 pieces of data?
                cmpi    512,R0            ;
                bne     keep_waiting      ;no then keep collecting data

                xor     R0,R0
                sti     R0,@fill_cnt      ;reset fill_cnt flag to wait
                ldi     @r1a_addr,R0      ;get address of data table 0
                ldi     @wfill,R1         ;get which table flag wfill
                not     R1,R1             ;not(wfill) to recover flag
                cmpi    0,R1              ;are we using table 0?
                beq     fft_00
                ldi     @r1b_addr,R0      ;no then get table 1 address
fft_00:
                sti     R0,@fft_addr      ;save location of fft when done

                ldp     @process_mode,DP ;set data page
```

```
                ldi     @cycle_cnt,R0    ;get cycle_cnt
                addi    1,R0             ;inc(cycle_cnt)
                sti     R0,@cycle_cnt    ;save cycle_cnt
                ldi     @cycle_cnt,R0    ;get current cycle_cnt
               ldi     @learn_cnt,R1    ;get count at start of learn cycle
                subi    R1,R0
               cmpi    300,R0           ;have we spent more than 500 cycles
                blt     keep_going       ; ~20 seconds
               cmpi    2,R5             ;if not timing burst then end mode
                blt     end_learning     ;
               cmpi    4,R5             ;if finished burst then not contin.
                bge     not_continuous   ;
                ldi     2,R0
                sti     R0,@cont_flag
                br      find_main_peak
not_continuous:
                ldi     1,R0
                sti     R0,@cont_flag
                br      find_main_peak

keep_going:
                call    find_amplitude  ;get average amplitude
                ldf     @test_ampl,R0
                cmpf    -1,R0           ;have we sampled background level
                bne     bg_ok
                ldf     @ave_ampl,R0    ;
                mpyf    1.5,R0          ;multiply by 1.5
               stf     R0,@test_ampl   ;save 1.5*first sample as background

bg_ok:
                ldi     *+AR4(IR1),R5   ;get waiting flag;
                cmpi    4,R5            ;have we finished with burst
                bge     no_fft_req      ;then just wait to ident interval

                cmpi    1,R5            ;
                blt     no_fft_req      ;

                xor     ENBL_GIE,ST     ;disable global interrupt
                call    setup_fft       ;calculate current fft

                xor     ENBL_GIE,ST     ;disable global interrupt
                call    fft_magnitude   ;calculate current fft magnitudes
                call    sum_ffts        ;add to total magnitudes
                or      ENBL_GIE,ST     ;enable global interrupt

no_fft_req:
;                xor     R0,R0
;                sti     R0,@fill_cnt    ;reset fill_cnt
               ldf     @ave_ampl,R0    ;get average signal amplitude
;                fix     R0,R7
;                call    write_queue
;                ldf     @min_ampl,R2    ;get minimum allowable amplitude
               ldf     @test_ampl,r2   ;use test background instead
;                fix     R2,R7
;                call    write_queue
              cmpf    R2,R0           ;if average amplitude < minlevel then
               blt     less_than_min
```

```
case3_0:        cmpi    0,R5            ; 0:begin
                bne     case3_1         ;
                ldi     @cycle_cnt,R0   ;
                sti     R0,*+AR3(IR1)   ;    save burst start time
                ldi     1,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=1
                br      case3_end       ;   end
case3_1:        cmpi    1,R5            ; 1:begin
                bne     case3_2         ;
                ldi     2,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=2
                br      case3_end       ;   end
case3_2:        cmpi    2,R5            ; 2:begin
                bne     case3_3         ;    do nothing
                br      case3_end       ;   end;
case3_3:        cmpi    3,R5            ; 3:begin
                bne     case3_4         ;
                ldi     2,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=2
                br      case3_end       ;   end
case3_4         cmpi    4,R5            ; 4:begin
                bne     case3_5         ;
                ldi     @cycle_cnt,R0   ;
                sti     R0,*+AR0(IR1)   ;    save interval time
                ldi     5,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=5
                br      case3_end       ;   end
case3_5         cmpi    5,R5            ; 5:begin
                bne     case3_end       ;
                ldi     6,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=6
                br      case3_end       ;   end
case3_end:      br      case4_end       ;end

less_than_min:
                                        ;case waiting of
case4_0:        cmpi    0,R5            ; 0:begin
                bne     case4_1         ;    do nothing
                br      case4_end       ;   end
case4_1:        cmpi    1,R5            ; 1:begin
                bne     case4_2         ;
                ldi     0,R0            ;
                sti     R0,*+AR3(IR1)   ;    reset burst start time
                sti     R0,*+AR4(IR1)   ;    waiting:=0
                call    clear_table     ;    clear FFT magnitude sums
                br      case4_end       ;   end
case4_2:        cmpi    2,R5            ; 2:begin
                bne     case4_3         ;
                ldi     @cycle_cnt,R0   ;
                sti     R0,*+AR2(IR1)   ;    save burst end time
                ldi     3,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=3
                br      case4_end       ;   end
case4_3:        cmpi    3,R5            ; 3:begin
                bne     case4_4         ;
                ldi     4,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=4
                br      case4_end       ;   end
```

```
case4_4:        cmpi    4,R5            ; 4:begin
                bne     case4_4         ;    do nothing
                br      case4_end       ;   end
case4_5:        cmpi    5,R5            ; 5:begin
                bne     case4_end       ;
                ldi     4,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=4
                br      case4_end       ;   end
case4_end:                              ;end

                ldi     *+AR4(IR1),R5   ;get waiting flag
                cmpi    6,R5            ;are we finished?
                bne     keep_waiting    ;no then keep waiting
                                        ;else

find_main_peak:
                xor     ENBL_GIE,ST     ;disable global interrupt
                call    copy_fft_sum    ;copy sum from temporary table
                call    average_fft_sum ;calculate max and average of sum
                call    find_peaks      ;find the peaks in sum table
                call    pick_main_peak  ;pick the largest peak
                or      ENBL_GIE,ST     ;enable global interrupt

save_template_info:

                ldi     @max_peak,R0    ;get frequency of maximum peak
                sti     R0,*+AR5(IR1)   ;save it in table
                ldi     0E100h,R7
                addi    R0,R7
                call    write_queue      ;send flag for learning mode

                ldi     *+AR3(IR1),R0   ;get burst start time
                ldi     *+AR2(IR1),R1   ;get burst end time
                subi    R0,R1           ;get difference
                ldi     *+AR3(IR1),R0   ;get burst start time
                ldi     *+AR0(IR1),R2   ;get interval time
                subi    R0,R2           ;get difference

               ldi     @t_bdur,AR2     ;get addr of template burst durration
               ldi     @t_int,AR3       ;get addr of template interval time

                ldi     @cont_flag,R0   ;get continous flag
                cmpi    1,R0
                blt     no_set_int
                ldi     0,R2
no_set_int:
                cmpi    2,R0
                blt     no_set_burst
                ldi     0,R1
no_set_burst:

                sti     R1,*+AR2(IR1)   ;save burst durration in template
                sti     R2,*+AR3(IR1)   ;save interval time in template
                ldi     0E200h,R7
                addi    R1,R7
                call    write_queue      ;send flag for learning mode
                ldi     0E300h,R7
                addi    R2,R7
```

```
                call    write_queue      ;send flag for learning mode

end_learning:
                ldi     0E010h,R7
                call    write_queue      ;send flag for learning mode
                xor     R0,R0
                sti     R0,@process_mode ;set for normal operation
                sti     R0,*+AR4(IR1)    ;waiting:=0 so it will detect later
                or      ENBL_GIE,ST      ;enable global interrupt

                pop     IR1
                pop     IR0
                pop     R7
                pop     R5
                pop     R4
                pop     R3
                pop     R2
                pop     R1
                pop     R0
                pop     AR7
                pop     AR6
                pop     AR5
                pop     AR4
                pop     AR3
                pop     AR2
                pop     AR1
                pop     AR0
                pop     DP

                retsu

;Find_amplitude finds the average absolute amplitude of the input signal
;to test for begining and end of signal burst.

find_amplitude:
                push    DP
                push    AR7
                push    R4
                push    R3
                push    R2
                push    R1
                push    R0
                push    RC
                push    RE

                ldf     0,R0
                ldi     @fft_addr,AR7    ;get address of current fft table
                ldi     511,RC           ;set to sum 255 words
                rptb    sum_average

                ldf     *AR7++,R1        ;get current input data
                absf    R1,R1            ;get absolute value
sum_average:    addf    R1,R0            ;add total

                float   512,R1           ;divide sum by 512 to get average
                call    FDIV             ;input absolute amplitude and
                stf     R0,@ave_ampl     ;save it in ave_ampl
```

```
                pop     RE
                pop     RC
                pop     R0
                pop     R1
                pop     R2
                pop     R3
                pop     R4
                pop     AR7
                pop     DP
                retsu

;Sum_ffts sums the current FFT magnitudes with those saved in the total
;fft magnitude table tmp_table

sum_ffts:
                push    DP
                push    AR7
                push    AR6
                push    R1
                push    R0
                push    RC
                push    RE

                ldf     0,R0
                ldi     @tmp_addr,AR7     ;get address of temporay table
                ldi     @fft_addr,AR6     ;get address of current fft table
                ldi     254,RC            ;set to sum 255 words
                rptb    sum_table

                ldp     @ram1_addr,DP     ;get data page of current fft table
                ldf     *AR6++,R0         ;get current fft magnitudes
                ldp     @tmp_table,DP     ;get data page of temporary table
                ldf     *AR7,R1           ;get fft sum from temporary table
                addf    R1,R0             ;add current fft to table
sum_table:      stf     R0,*AR7++         ;and save sum in tmp_table

                pop     RE
                pop     RC
                pop     R0
                pop     R1
                pop     AR6
                pop     AR7
                pop     DP
                retsu

;Copy_fft_sum copys Sum of FFT magnitudes back to the FFT arrays so that
;Find_peaks subroutine can operate

copy_fft_sum:
                push    DP
                push    AR7
                push    AR6
                push    R0
                push    RC
                push    RE

                ldi     @tmp_addr,AR7     ;get address of temporay table
                ldi     @fft_addr,AR6     ;get address of current fft table
```

```
                ldi     254,RC              ;set to copy 255 words
                rptb    copy_sum

                ldp     @tmp_table,DP       ;get data page of temporary table
                ldf     *AR7++,R0           ;get fft sum from temporary table
                ldp     @ram1_addr,DP       ;get data page of current fft table
copy_sum:       stf     R0,*AR6++           ;fft_table[AR6]:=tmp_table[AR7]

                pop     RE
                pop     RC
                pop     R0
                pop     AR6
                pop     AR7
                pop     DP
                retsu

;Find the average amplitude of fft peaks from the sum of all the ffts

average_fft_sum:
                push    DP
                push    RE
                push    RC
                push    AR5
                push    AR6
                push    R0
                push    R1
                push    R2
                push    R3
                push    R4
                push    IR0
                ldp     @r1a_addr,DP        ;set data page
                xor     R0,R0
                stf     R0,@max_fft         ;max_fft:=0
                stf     R0,@ave_fft         ;ave_fft:=0
                ldi     @fft_addr,AR5       ;get address of fft table
                ldi     1,IR0               ;set index to 1
                ldi     254,RC              ;255 values
                rptb    end_ave_loop
                cmpi    58,IR0              ;if less than 900HZ then
                                            ;@ sample rate 8000Hz
                blt     end_ave_loop        ;don't calculate max or ave
                ldf     *+AR5(IR0),R0       ;get fft_real[IR0]
                ldf     @max_fft,R1         ;get current max_fft
                cmpf    R1,R0               ;if max_fft>new magnitude
                blt     no_new_amax         ;then don't pdate max_fft
                stf     R0,@max_fft         ;else store new max
no_new_amax:
                ldf     @ave_fft,R1
                addf    R0,R1
                stf     R1,@ave_fft         ;ave_fft:=ave_fft+max_fft
end_ave_loop:   addi    1,IR0               ;inc(IR0)

                ldf     @ave_fft,R0         ;get average total
                ldf     197,R1              ;get number of additions
                                            ;note 197 for sample rate
                                            ;of 8000 Hz
                call    FDIV
                stf     R0,@ave_fft         ;save it
```

```
ave_sum_end:
                pop     IR0
                pop     R4
                pop     R3
                pop     R2
                pop     R1
                pop     R0
                pop     AR6
                pop     AR5
                pop     RC
                pop     RE
                pop     DP
                retsu

;Pick_main_peak scans the peak table generated by find_peaks and looks
;for the peak with the greatest magnitude

pick_main_peak:
                push    DP
                push    AR5
                push    AR6
                push    AR7
                push    R0
                push    R1
                push    RC
                push    RE
                ldf     0,R0                    ;test_mag:=0
                ldi     0,AR5                   ;max_peak:=0
                ldp     @peak_hz,DP             ;set data page
                ldi     @peak_table,AR7         ;get address of peak table
                ldi     @peak_mtable,AR6        ;get address of peak max table
                ldi     @peak_cnt,RC            ;get peak_cnt
                subi    1,RC                    ;set to test all peaks found
                rptb    next_peak

                ldf     *AR6++,R1               ;get peak magnitude
                cmpf    R1,R0                   ;is it bigger than current
                bgt     next_peak               ;if greater than
                ldf     R1,R0                   ;then save magnitude and
                ldi     AR7,AR5                 ;index to peak frequency table

next_peak:      addi    1,AR7                   ;inc(index)


                ldi     *AR5,R0                 ;get frequency of largest peak
                sti     R0,@max_peak            ;save it in max_peak
                pop     RE
                pop     RC
                pop     R1
                pop     R0
                pop     AR7
                pop     AR6
                pop     AR5
                pop     DP
                retsu

;Clear_table clears the FFT magnitude sum table
```

```
clear_table:
                push    DP
                push    AR7
                push    R0
                push    RC
                push    RE

                ldf     0,R0
                ldp     @tmp_addr,DP    ;get data page of temporary table ptr
                ldi     @tmp_addr,AR7   ;get address of temporay table
                ldp     @tmp_table,DP   ;get data page of temporary table
                ldi     254,RC          ;set to clear 255 words
                rptb    clr_table

clr_table:      stf     R0,*AR7++       ;tmp_table[AR7]:=0.0

                pop     RE
                pop     RC
                pop     R0
                pop     AR7
                pop     DP
                retsu

;Matching Subroutine matches peaks found in find_peaks to info
;stored in templates

find_match:
                push    DP
                push    AR0             ;template interval time
                push    AR1             ;template burst duration
                push    AR2             ;burst timer end times
                push    AR3             ;burst timer start times
                push    AR4             ;waiting flags
                push    AR5             ;template peak values
                push    AR6             ;peak table magnitudes
                push    AR7             ;peak table peak values
                push    R0
                push    R1
                push    R2              ;min_power
                push    R3              ;2.5*ave_fft
                push    R4              ;foung flag
                push    R5              ;waiting flag
                push    R7              ;
                push    IR0             ;peak tables index
                push    IR1             ;template tables index

                ldp     @peak_hz,DP     ;set data page
                ldi     @peak_table,AR7 ;get address of peak table
                ldi     @peak_mtable,AR6 ;get address of peak max table
                ldi     @t_rp,AR5       ;get address of template peaks
                ldi     @t_wait,AR4     ;get address of template flags
                ldi     @t_start,AR3    ;get address of burst timer start
                ldi     @t_end,AR2      ;get address of burst timer end
                ldi     @t_intc,AR0     ;get address of interval timer

                ldf     @min_power,R2
                ldf     @ave_fft,R3
;                mpyf    2.5,R3          ;R1:=2.5*ave_fft
```

```
                mpyf    6.25,R3          ;R1:=6.25*ave_fft no sqrts

                ldi     0,IR1            ;template_index:=0
template_loop:
                ldi     0,R4             ;found:=false
                ldi     0,IR0            ;peak index:=0
                ldi     @peak_cnt,RC     ;get number of peaks found
                subi    1,RC
                rptb    test_temps
                ldf     *+AR6(IR0),R0    ;get next peak in peak table
                cmpf    R3,R0            ;if peak_mag<2.5*ave_fft
                blt     test_temps       ;then not a valid peak
                cmpf    R2,R0            ;if peak_mag<min_power limit
                blt     test_temps       ;then not a valid peak

                float   *+AR5(IR1),R1    ;get template value
                float   *+AR7(IR0),R0    ;R0:=peak frequency

                subf    R0,R1            ;R1:=R1-R0
                absf    R1,R1
                cmpf    3,R1             ;if abs(peak-template)>tolerance
                bgt     test_temps       ;then no peak match
                addi    1,R4             ;else found:=true
test_temps:     addi    1,IR0            ;inc(peak_index)
                ldi     *+AR4(IR1),R5    ;get waiting flag
                cmpi    0,R4             ;if not(found) then
                beq     not_found        ;no template match

                                         ;case waiting of
case1_0:        cmpi    0,R5             ; 0:begin
                bne     case1_1          ;
                ldi     @cycle_cnt,R0    ;    get current cycle count
                sti     R0,*+AR3(IR1)    ;    temp[IR1].burst_start:=cycle_cnt
                ldi     1,R0             ;
                sti     R0,*+AR4(IR1)    ;    waiting:=1
                br      case1_end        ;   end
case1_1:        cmpi    1,R5             ; 1:begin
                bne     case1_2          ;
                ldi     2,R0             ;
                sti     R0,*+AR4(IR1)    ;    waiting:=2
                br      case1_end        ;   end
case1_2:        cmpi    2,R5             ; 2:begin
                bne     case1_3          ;    do nothing just wait


                ldi     @t_bdur,AR1      ;    get address of template burst
dur
                ldi     *+AR1(IR1),R0    ;    get template burst durration
                cmpi    0,R0
                bne     no_test_cont     ;    else dont issue message
test_cont:
                ldi     @cycle_cnt,R0    ;    get burst end time
                ldi     *+AR3(IR1),R1    ;    get burst start time
                subi    R1,R0            ;    burst durration:=end-start
                 ldi     31,R1           ;     get continuous burst dur. (2
sec)
                subi    R1,R0
                float   R0,R0            ;    convert to floating point
```

```
                absf    R0,R0           ;    get |R0-R1|
                float   31,R1           ;    R1 set for +-15% tolerance
                mpyf    0.30,R1         ;      on burst duration
                cmpf    R0,R1           ;    if difference between burst
                                        ;     duration and stored value
                                        ;     in template < 15% of stored
                                        ;     value then burst matches
                blt     no_test_cont    ;

                ldi     0,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=0

                ldi     0C000H,R7       ;    diplay message B00 plus index
                addi    IR1,R7          ;

                push    AR4             ;
                push    IR0             ;
                ldi     @peak_ctable,AR4;   store message in peak_table
                ldi     @peaks_tcnt,IR0 ;
                sti     R7,*+AR4(IR0)   ;
                addi    1,IR0
                cmpi    10,IR0          ;
                bne     nov4
                xor     IR0,IR0
nov4:
                sti     IR0,@peaks_tcnt ;
                pop     IR0             ;
                pop     AR4             ;

no_test_cont:
                br      case1_end       ;   end;
case1_3:        cmpi    3,R5            ; 3:begin
                bne     case1_4         ;
                ldi     2,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=2
                br      case1_end       ;   end
case1_4:        cmpi    4,R5            ; 4:begin
                bne     case1_end       ;
                ldi     2,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=2
                                        ;   end
case1_end:      br      temp_loop_end   ;end

not_found:

                                        ;case waiting of
case2_0:        cmpi    0,R5            ; 0:begin
                bne     case2_1         ;    do nothing
                br      case2_end       ;   end
case2_1:        cmpi    1,R5            ; 1:begin
                bne     case2_2         ;
                ldi     4,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=4
                br      case2_end       ;   end
case2_2:        cmpi    2,R5            ; 2:begin
                bne     case2_3         ;
                ldi     @cycle_cnt,R0   ;    get current cycle count
               sti     R0,*+AR2(IR1)    ;    temp[IR1].burst_end:=cycle_cnt
```

```
                ldi     3,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=3
                br      case2_end       ;   end
case2_3:        cmpi    3,R5            ; 3:begin
                bne     case2_4         ;
                ldi     0,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=0

test_burst:
                ldi     *+AR2(IR1),R0   ;    get burst end time
                ldi     *+AR3(IR1),R1   ;    get burst start time
                subi    R1,R0           ;    burst durration:=end-start
                ldi     @t_bdur,AR1     ;    get address of template burst
dur
                ldi     *+AR1(IR1),R1   ;    get template burst durration
                subi    R1,R0
                float   R0,R0           ;    convert to floating point
                absf    R0,R0           ;    get |R0-R1|
                float   *+AR1(IR1),R1   ;    R1 set for +-15% tolerance
                mpyf    0.30,R1         ;      on burst duration
                cmpf    R0,R1           ;    if difference between burst
                                        ;     duration and stored value
                                        ;     in template < 15% of stored
                                        ;     value then burst matches
                blt     no_message      ;    else dont issue message
found_burst:    ldi     0B000H,R7       ;    diplay message B00 plus index
                addi    IR1,R7          ;

                push    AR4             ;
                push    IR0             ;
                ldi     @peak_ctable,AR4;   store message in peak_table
                ldi     @peaks_tcnt,IR0 ;
                sti     R7,*+AR4(IR0)   ;
                addi    1,IR0
                cmpi    10,IR0          ;
                bne     nov1
                xor     IR0,IR0
nov1:
                sti     IR0,@peaks_tcnt ;
                pop     IR0             ;
                pop     AR4             ;

test_interval:
                ldi     @t_int,AR1      ;    get address of template interval
time
                float   *+AR3(IR1),R0   ;    get burst start time
                float   *+AR0(IR1),R1   ;    get current interval timer cnt
                cmpf    0,R1            ;    if interval_timer = 0 then
                beq     not_zero        ;     not currently timing interval

                subf    R1,R0,R1        ;    R1:=interval_timer-burst_start
                float   *+AR1(IR1),R0   ;    get template interval time
                subf    R0,R1           ;
                absf    R1,R1           ;    get magnitude of difference
                float   *+AR1(IR1),R0   ;    get template interval time
                mpyf    0.10,R0         ;    set 10% tolerance
                cmpf    R0,R1           ;   if magn of diff < 10% of stored
;               bgt     no_message      ;     interval time then interval
```

```
match
                bgt     not_zero        ;     interval time then interval
match
                ldi     0,R0
                sti     R0,*+AR0(IR1)   ;    interval_timer:=0;
found_int:      ldi     0B100H,R7       ;    diplay message B10 plus index
                addi    IR1,R7          ;

                push    AR4             ;
                push    IR0             ;
                ldi     @peak_ctable,AR4;   store message in peak_table
                ldi     @peaks_tcnt,IR0  ;
                sti     R7,*+AR4(IR0)   ;
                addi    1,IR0           ;
                cmpi    10,IR0          ;
                bne     nov2
                xor     IR0,IR0
nov2:
                sti     IR0,@peaks_tcnt  ;
                pop     IR0             ;
                pop     AR4             ;

not_zero:
                ldi     *+AR3(IR1),R0
                sti     R0,*+AR0(IR1)   ;    interval_timer:=burst_start
                br      case2_end
no_message:
                ldi     0,R7
                not     R7,R7

                br      case2_end       ;   end
case2_4:        cmpi    4,R5            ; 4:begin
                bne     case2_end       ;
                ldi     0,R0            ;
                sti     R0,*+AR4(IR1)   ;    waiting:=0
                                        ;   end
case2_end:      br      temp_loop_end   ;end

temp_loop_end:  addi    1,IR1           ;inc(template_index)
                cmpi    5,IR1           ;more templates?
                blt     template_loop   ;compare next template
                                        ;else we are done
                pop     IR1
                pop     IR0
                pop     R7
                pop     R5
                pop     R4
                pop     R3
                pop     R2
                pop     R1
                pop     R0
                pop     AR7
                pop     AR6
                pop     AR5
                pop     AR4
                pop     AR3
                pop     AR2
                pop     AR1
```

```
                pop     AR0
                pop     DP

end_match:      retsu

;Subroutine to find peaks in spectrum from FFT
find_peaks:
                push    DP
                push    AR4
                push    AR5
                push    AR6
                push    AR7
                push    R0
                push    R1
                push    R2
                push    R3
                push    IR0
                push    IR1
                ldp     @peak_hz,DP             ;set data page
                ldi     @peak_table,AR7         ;get address of peak table
               ldi     @peak_mtable,AR6        ;get address of peak max table
                xor     R0,R0
                sti     R0,@peak_cnt            ;peak_cnt:=0
                sti     R0,@g_up                ;g_up:=0
                ldi     R0,IR0                  ;peak_hz table index:=0
                ldi     R0,IR1                  ;fft peak table index:=0
                ldf     0,R1
                ldi     49,RC
                rptb    init_peak
                stf     R1,*AR6++               ;set all peak maximums to 0
init_peak:      sti     R0,*AR7++               ;set all peaks to 0
                ldi     @peak_table,AR7         ;get address of peak table
               ldi     @peak_mtable,AR6       ;get address of peak max table
                ldi     @fft_addr,AR5           ;get current fft peak table
                ldi     @g_up,R2                ;get g_up flag

                ldi     254,RC
                rptb    test_peaks
                cmpi    57,IR1                  ;if less than 900HZ
                                                ;@ sample rate 8000Hz
                blt     test_peaks              ;dont look for peaks
                ldf     *+AR6(IR0),R0           ;get current peak_max[IR0]
                ldf     *+AR5(IR1),R1           ;get current fft_value[IR1]
                cmpf    R0,R1                   ;if R1>R0 (ie fft>peak max)
               ble     going_down1             ;then we are going down a peak
                cmpi    1,R2                    ;if g_up<1 then
                bge     going_up1               ;
                addi    1,R2                    ;inc(g_up)
going_up1:
               stf     R1,*+AR6(IR0)           ;peak_max[IR0]:=fft_value[IR1];
                sti     IR1,*+AR7(IR0)          ;peak_hz[IR0]:=[IR1]
                br      test_peaks
going_down1:
                cmpi    0,R2                    ;if g_up>0 then
                ble     test_peaks              ;begin
                subi    1,R2                    ; dec(g_up)
                cmpi    0,R2                    ; if g_up=0
                bne     going_down3             ; begin
```

```
                ldf     @max_fft,R3             ;
;               mpyf    0.25,R3                 ;
                mpyf    0.0625,R3               ;  no sqrt
                ldf     *+AR6(IR0),R0           ;  get peak_max[IR0]
                cmpf    R3,R0                   ;  if peak_max[IR0]>0.25*max_fft
                ble     going_down4             ;
                addi    1,IR0                   ;   inc(peak_cnt)
                sti     IR0,@peak_cnt           ;   save peak_cnt
                br      test_peaks
going_down4:    ldf     0,R0                    ;  else
                stf     R0,*+AR7(IR0)           ;   peak_max[IR0]:=0;
                xor     R0,R0                   ;
                sti     R0,*+AR6(IR0)           ;   peak_hz[IR0]:=0;
                                                ; end
going_down3:                                    ;end;
test_peaks:     addi    1,IR1                   ;inc(IR1)

;               ldi     @peak_cnt,R0            ;get current peak count
;                ldi     @peak_ctable,AR4       ;get address of peak count
table
;               ldi     @cycle_cnt,IR0          ;get current cycle
;               sti     R0,*+AR4(IR0)

                pop     IR1
                pop     IR0
                pop     R3
                pop     R2
                pop     R1
                pop     R0
                pop     AR7
                pop     AR6
                pop     AR5
                pop     AR4
                pop     DP
end_find_peaks:
                retsu

;Realtime Data array filling routine called by interrupt
fill_array:
                nop
                nop
                push    ST
                push    DP
                push    AR7
                push    R0
                push    IR0
                push    R7
                ldp     @r1a_addr,DP            ;set data page
                ldi     @r1a_addr,AR7           ;get address of table 0
                ldi     @wfill,R0               ;get table filling flag
                cmpi    0,R0                    ;are we filling table 0 ?
                beq     fill0                   ;yes then just continue
                ldi     @r1b_addr,AR7           ;else get address of table 1
fill0:
                call    receive0
;               call    get_data                ;get the current realtime data
                ldi     @fcnt,IR0               ;get table index
                stf     R7,*+AR7(IR0)           ;store data in table
```

```
                addi    1,IR0                   ;inc index
                cmpi    512,IR0                 ;is table full?
                bne     no_reset_cnt            ;no then keep filling
                sti     IR0,@fill_cnt           ;else set fill_cnt flag
                ldi     @wfill,R0               ;get table filling flag
                not     R0,R0                   ;invert which table flag
                sti     R0,@wfill               ;save which table flag
                ldi     0,IR0                   ;reset index

no_reset_cnt:
                sti     IR0,@fcnt              ;save index
                pop     R7
                pop     IR0
                pop     R0
                pop     AR7
                pop     DP
                pop     ST
                nop
                nop
end_fill:
                reti
                rets

;Place holder subroutine for getting realtime data
;returns data in R7

get_data:
                push    DP                      ;save data page
                push    AR7                     ;save AR7
                ldp     @data_index,DP          ;get current real data page
               ldi     @data_index,AR7         ;get current real data address
                ldf     *AR7++,R7               ;get data, inc(data_index)
                 subf    128,R7                   ;subtract 128 for 8 bit voc
format
                sti     AR7,@data_index         ;save new index
                pop     AR7                     ;restore registers
                pop     DP
                retsu

;Place holder for do nothing interrupt routine
null_int:       retiu


;Subrotuine for calculating FFT magnitudes, maximum and average
fft_magnitude:
                push    DP
                push    RE
                push    RC
                push    AR5
                push    AR6
                push    R0
                push    R1
                push    R2
                push    R3
                push    R4
                push    IR0
                ldp     @r1a_addr,DP            ;set data page
                xor     R0,R0
```

```
                stf     R0,@max_fft         ;max_fft:=0
                stf     R0,@ave_fft         ;ave_fft:=0
                ldi     @fft_addr,AR5       ;get address of fft table
                ldi     @fft_addr,AR6
               addi    512,AR6             ;get address of end of fft table
                ldi     1,IR0               ;set index to 1
                ldi     254,RC              ;255 values
                rptb    end_magn_loop
                cmpi    58,IR0              ;if less than 900HZ then
                                            ;@ sample rate 8000Hz
                blt     end_magn_loop       ;don't calculate max or ave
                ldf     *+AR5(IR0),R0       ;get fft_real[IR0]
                ldf     *-AR6(IR0),R1       ;get fft_imag[IR0]
                mpyf    R0,R0               ;square real part
                mpyf    R1,R1               ;square imag part
                addf    R1,R0               ;sum the squares
               ; call    SQRT                ;take the sqrt
               stf     R0,*+AR5(IR0)       ;save it in table at real value
                ldf     @max_fft,R1         ;get current max_fft
                cmpf    R1,R0               ;if max_fft>new magnitude
                blt     no_new_max          ;then don't pdate max_fft
                stf     R0,@max_fft         ;else store new max
no_new_max:
                ldf     @ave_fft,R1
                addf    R0,R1
                stf     R1,@ave_fft         ;ave_fft:=ave_fft+max_fft
end_magn_loop:  addi    1,IR0               ;inc(IR0)

                ldf     @ave_fft,R0         ;get average total
                ldf     197,R1              ;get number of additions
                                            ;note 197 for sample rate
                                            ;of 8000 Hz
                call    FDIV
                stf     R0,@ave_fft         ;save it
fft_mag_end:
                pop     IR0
                pop     R4
                pop     R3
                pop     R2
                pop     R1
                pop     R0
                pop     AR6
                pop     AR5
                pop     RC
                pop     RE
                pop     DP
                retsu

*************************************************************************
*       aicreset                                                        *
*                                                                       *
*       Reset and intialize the AIC                                     *
*                                                                       *
*       Operations:  Set up timer 0 to supply AIC master clock          *
*                Reset the AIC                                          *
*                Intial the serial ports                                *
*                Take AIC out of reset                                  *
*                Intialize the AIC                                      *
```

```
*               Enable receive interrupts                    *
*                                                            *
**************************************************************

aicreset:
        ldi         2,iof           ;xf0 to output, set xf0 to 0
        ldi          @t0_ctladdr,ar0      ;get address of timer control
register
        ldi         1,r1            ;tclk0 will equal h1/2
        sti         r1,*+ar0(8)           ;set the period register to 1
        ldi         @t0_ctlinit,r1      ;get timer 0 setup value
        sti         r1,*ar0         ;set timer 0 to run in pulse mode

        ldi         @p0_addr,ar0       ;get address of serial port 0
        ldi         111h,r1
        sti         r1,*+ar0(2)          ;intialize transmit port control
        sti         r1,*+ar0(3)          ;intialize receive port control
        ldi         @p0_global,r1       ;intialize port 0 global control
        sti         r1,*ar0
        xor         r1,r1
        sti         r1,*+ar0(8)           ;set transmit data to 0

        ldi     0,R7
        rpts    99
        nop                         ;wait for 50 timer out clocks
        ldi         6,iof           ;set xf0 to 1, !reset AIC
                            ;set up the aic
        call    wait_transmit_0     ;poll for transmit interrupt
        ldi         3,r1
        sti         r1,*+ar0(8)           ;secondary transmittion
        call    wait_transmit_0
        ldi         1a34h,r1        ;set the sampling rate
        sti         r1,*+ar0(8)
        ldi         *+ar0(12),r1

;       call    wait_transmit_0     ;poll for transmit interrupt
;       ldi         3,r1
;       sti         r1,*+ar0(8)           ;secondary transmittion
;       call    wait_transmit_0
;       ldi         3872h,r1        ;set the sampling rate for 10.3kHz
;       ldi         3C7Ah,r1        ;set the sampling rate for 9.6kHz
;       ldi         346Ah,r1        ;set the sampling rate for 11kHz
;       sti         r1,*+ar0(8)
;       ldi         *+ar0(12),r1

        call    wait_transmit_0     ;setup aic transmit and recieve
        ldi         3,r1            ;sampling rates
        sti         r1,*+ar0(8)
        call    wait_transmit_0
        ldi         2a7h,r1
        sti         r1,*+ar0(8)
        ldi         *+ar0(12),r1
        xor         if,if           ;clear out all interrupt flags
          xor     ENBL_XINT0,IE       ;disable serial port 0 tx int
        or    @enbl_sp0_r,ie      ;enable serial port 0
        rets

;wait_transmit_0:
```

```
;       xor         if,if           ;wait for the transmit interrupt
;wloop:       tstb    10h,if         ;flag to be set.
;       bz    wloop
;       rets

wait_transmit_0:
wloop:       tstb    1h,R7          ;flag to be set.
        bz    wloop
        ldi     0,R7
        rets


transmit0: ldi     1,R7               ;
          reti

transmit1: reti
recieve1:  reti
timer1:    reti

receive0:;  push    st              ;save registers
        push    r0
        push    ar0
        push    dp

        ldp         PARMS
        ldi         @p0_addr,ar0        ;get port address
        ldi         *+ar0(12),r0        ;read input
          ldi     R0,R7
          lsh     16,R7
          ash     -18,R7
          float   R7,R7
        sti         r0,*+ar0(8)           ;send output
        pop         dp              ;restore registers
        pop         ar0
        pop         r0
       ; pop         st
          rets
        reti

*****************************************************************
*  hread()                                                      *
*                                                               *
*  Read data from c30 memory and write to communications reg.     *
*                                                               *
*****************************************************************
hread16:
        push    st                  ;save registers
        push    r0
        push    ar6
          push    IR0
        push    dp

        ldp         process_mode,DP
          ldi     @peak_ctable,AR6    ;get address of output buffer
          ldi     @peaks_hcnt,IR0     ;get pointer to head of buffer
          cmpi    @peaks_tcnt,IR0     ;get pointer to tail of buffer
          beq     send_zero           ;if equal do nothing
          ldi     *+AR6(IR0),R0       ;else get nest message
```

```
          ldi         @hostport,ar6       ;load host port address
          sti         r0,*ar6           ;store data
            addi    1,IR0                ;inc head pointer
            cmpi    10,IR0               ;have we rolled over
            bne     miss_reset_pointers ;no then save new head pointer

            xor     IR0,IR0             ;else reset head and tail so buffer
            sti     IR0,@peaks_hcnt     ;doesn;t overflow (10 words max)
;            sti     IR0,@peaks_tcnt
            br      no_reset_pointers
miss_reset_pointers:
            sti     IR0,@peaks_hcnt     ;save head pointer
no_reset_pointers:
          pop         dp                ;restore registers
            pop     IR0
          pop         ar6
          pop         r0
          pop         st
          rets
send_zero:
            xor     R0,R0
          ldi         @hostport,ar6       ;load host port address
          sti         r0,*ar6           ;store data
          pop         dp                ;restore registers
            pop     IR0
          pop         ar6
          pop         r0
          pop         st
          rets

;command write

cmd_write:
          push    st                    ;save registers
          push    r0
          push    ar6
          push    dp
            push    R7

;         ldp         process_mode,DP
            xor     R0,R0
          ldi         @hostport,ar6       ;load host port address
          ldi         *ar6,R7             ;get input command
            cmpi    0,R7
            beq     no_echo
            call    write_queue             ;echo to output queue
            cmpi    1,R7
            bne     not_1
            not     R0,R7
            sti     R7,@process_mode    ;set flag for learning mode
not_1:
no_echo:
            call    hread16             ;write next queue entry to PC

            pop     R7
          pop         dp                ;restore registers
          pop         ar6
          pop         r0
```

```
        pop         st
        reti

write_queue:
                push    AR4             ;
                push    IR0             ;
                ldi     @peak_ctable,AR4 ;   store message in output queue
                ldi     @peaks_tcnt,IR0 ;
                sti     R7,*+AR4(IR0)   ;
                addi    1,IR0           ;
                cmpi    10,IR0          ;
                bne     nov3
                xor     IR0,IR0
nov3:
                sti     IR0,@peaks_tcnt ;
                pop     IR0             ;
                pop     AR4             ;
                rets

                end.
```

```
*******************************************************************************
*
*
* File : warnmath.asm
*
*
*
*******************************************************************************
*
*************************************************************
*   SUBROUTINE: FPINV                                       *
*                                                           *
*   WRITTEN BY: GARY A. SITTON                              *
*               GAS LIGHT SOFTWARE                          *
*               HOUSTON, TEXAS                              *
*               MARCH 1989.                                 *
*                                                           *
*   FLOATING POINT INVERSE: R0 <= 1/R0                      *
*                                                           *
*   APPROXIMATE ACCURACY: 8 DECIMAL DIGITS.                 *
*   INPUT RESTRICTIONS: R0 != 0.0.                          *
*   REGISTERS FOR INPUT: R0.                                *
*   REGISTERS USED AND RESTORED: DP AND SP.                 *
*   REGISTERS ALTERED: R0-2 AND R4.                         *
*   REGISTERS FOR OUTPUT: R0.                               *
*   ROUTINES NEEDED: NONE.                                  *
*   EXECUTION CYCLES (MIN, MAX): 33 , 33.                   *
*************************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  FPINV

;       INTERNAL CONSTANTS

        .DATA

ONE     .SET    1.0
TWO     .SET    2.0

MSK     .WORD   0FF7FFFFFH

;        .TEXT
         .sect  ".mtext"

;       START OF FPINV PROGRAM

FPINV:

        LDF     R0,R0           ; TEST F
        RETSZ                   ; RETURN NOW IF F = 0

;       GET APPROXIMATION TO 1/F.  FOR F = (1+M) * 2**E
;       AND 0 <= M < 1, USE: X[0] = (1-M/2) * 2**-E

        PUSH    DP              ; SAVE DATA PAGE POINTER
```

```
        LDP     @MSK            ; LOAD DATA PAGE POINTER
        PUSHF   R0              ; SAVE AS FLT. PT. F = (1+M) * 2**E
        POP     R1              ; FETCH BACK AS INTEGER
        XOR     @MSK,R1         ; COMPLEMENT E & M BUT NOT SIGN BIT
        PUSH    R1              ; SAVE AS INTEGER, AND BY MAGIC...
        POPF    R1              ; R1 <= X[0] = (1-M/2) * 2**-E.
        POP     DP              ; UNSAVE DP

;       NEWTON ITERATION FOR: Y(X) = X - 1/F = 0 ...

        MPYF    R1,R0,R4        ; R4 <= F * X[0]
        SUBRF   TWO,R4          ; R4 <= 2 - F * X[0]
        MPYF    R4,R1           ; R1 <= X[1] = X[0] * (2 - F * X[0])

        MPYF    R1,R0,R4        ; R4 <= F * X[1]
        SUBRF   TWO,R4          ; R4 <= 2 - F * X[1]
        MPYF    R4,R1           ; R1 <= X[2] = X[1] * (2 - F * X[1])

        MPYF    R1,R0,R4        ; R4 <= F * X[2]
        SUBRF   TWO,R4          ; R4 <= 2 - F * X[2]
        MPYF    R4,R1           ; R1 <= X[3] = X[2] * (2 - F * X[2])

;       FOR THE LAST ITERATION: X[4] = (X[3] * (1 - (F * X[3]))) + X[3]

        RND     R0,R4           ; ROUND F BEFORE LAST MULTIPLY
        RND     R1,R0           ; ROUND X[3] BEFORE MULTIPLIES
        MPYF    R0,R4           ; R4 <= F * X[3] = 1 + EPS

;       FINISH ITERATION AND RETURN

        POP     R2              ; R2 <= RETURN ADDRESS
        BUD     R2              ; RETURN (DELAYED)
        SUBRF   ONE,R4          ; R4 <= 1 - F * X[3] = EPS
        MPYF    R0,R4           ; R4 <= X[3] * EPS
       ADDF    R4,R1,R0        ; R0 <= X[4] = (X[3]*(1 - (F*X[3]))) + X[3]

*************************************************************
*   SUBROTUINE: FDIV                                        *
*                                                           *
*   WRITTEN BY: GARY A. SITTON                              *
*               GAS LIGHT SOFTWARE                          *
*               HOUSTON, TEXAS                              *
*               APRIL 1989.                                 *
*                                                           *
*   FLOATING POINT DIVIDE FUNCTION: R0 <= R0/R1.            *
*                                                           *
*   APPROXIMATE ACCURACY: 8 DECIMAL DIGITS.                 *
*   INPUT RESTRICTIONS: R1 != 0.0.                          *
*   REGISTERS FOR INPUT: R0 (DIVIDEND) AND R1 (DIVISOR).    *
*   REGISTERS USED AND RESTORED: DP AND SP.                 *
*   REGISTERS ALTERED: R0-4.                                *
*   REGISTERS FOR OUTPUT: R0 (QUOTIENT).                    *
*   ROUTINES NEEDED: FPINV.                                 *
*   EXECUTION CYCLES (MIN, MAX): 43 , 43.                   *
*************************************************************

;       EXTERNAL PROGRAM NAMES
```

```
        .GLOBL  FDIV
        .GLOBL  FPINV

;        .TEXT
         .sect  ".mtext"

;       START OF FDIV PROGRAM

FDIV:

        RND     R0,R3           ; R3 <= RND X
        LDF     R1,R0           ; R1 <= Y
        CALL    FPINV           ; R0 <= 1/Y
        RND     R0              ; ROUND BEFORE *
        MPYF    R3,R0           ; R0 <= X

        RETS                    ; RETURN

        .END
*
 *
*
 *
*************************************************************************
****
```

```
*****************************************************************************
****
* FILENAME   : warnfft.asm
*
* ADAPTED FROM     : Texas Instrument
*
* DATE             : 23rd June 1994
*
* VERSION    : 3.0
*
*****************************************************************************
****
* VER   DATE             COMMENTS
* --- ------------
-----------------------------------------------------
* 1.0        18th July 91       Original Release.
* 2.0    23rd July 91     Most Stages Modified.
*                  Minimum FFT Size increased from 32 to 64.
*                  Faster in place bit reversing algorithm.
*                  Program size increased by about 100 words.
*                  One extra data word required.
* 3.0    23rd June 94     Used as 2 512 word FFT banks
*                         Both Bank2 to reside in RAM1 internal bank
*                         Code to reside in RAM0 intenal bank
*                         NOT called as a C subroutine
*                         Called as an assembly language routine only!
*
*****************************************************************************
****
*
* SYNOPSIS: int    ffft_rl( FFT_SIZE, LOG_SIZE, SOURCE_ADDR, DEST_ADDR,
*                           SINE_TABLE, BIT_REVERSE );
*
*           int    FFT_SIZE     ; 64, 128, 256, 512, 1024, ...
*           int    LOG_SIZE     ;  6,   7,   8,   9,   10, ...
*           float *SOURCE_ADDR       ; Points to location of source data.
*           float *DEST_ADDR   ; Points to where data will be
*                              ; operated on and stored.
*           float *SINE_TABLE  ; Points to the SIN/COS table.
*           int    BIT_REVERSE ; =  0, Bit Reversing is disabled.
*                              ; <> 0, Bit Reversing is enabled.
*
*           NOTE: 1) If SOURCE_ADDR = DEST_ADDR, then in place bit
*                    reversing is performed, if enabled (more
*                    processor intensive).
*                 2) FFT_SIZE must be >= 64 (this is not checked).
*
* DESCRIPTION:     Generic function to do a radix-2 FFT computation on the
C30.
*           The data array is one of 2 FFT_SIZE-long tables with only
*               real data. The output is stored in the same locations with
*                real and imaginary points R and I as follows:
*
*                  DEST_ADDR[0]              -> R(0)
*                                          R(1)
*                                          R(2)
*                                          R(3)
```

```
*                                       .
*                                         .
*                                       R(FFT_SIZE/2)
*                                         I(FFT_SIZE/2 - 1)
*                                       .
*                                         .
*                                       I(2)
*               DEST_ADDR[FFT_SIZE - 1] -> I(1)
*
*          This subroutine is based on the FORTRAN program in the paper
*          by Sorensen et al., June 1987 issue of Trans. on ASSP.
*
*          Bit reversal is optionally implemented at the beginning
*          of the function.
*
*          The sine/cosine table for the twiddle factors is expected
*          to be supplied in the following format:
*
*          SINE_TABLE[0]                -> sin(0*2*pi/FFT_SIZE)
*                                     sin(1*2*pi/FFT_SIZE)
*                                                .
*                                                .
*                                     sin((FFT_SIZE/2-2)*2*pi/FFT_SIZE)
*          SINE_TABLE[FFT_SIZE/2 - 1] -> sin((FFT_SIZE/2-1)*2*pi/FFT_SIZE)
*
*          NOTE: The table is the first half period of a sine wave.
*
*          Stack structure upon call:
*
*                        +-------------+
*          -FP(7)     | BIT_REVERSE |
*          -FP(6)     | SINE_TABLE  |
*          -FP(5)     | DEST_ADDR   |
*          -FP(4)     | SOURCE_ADDR |
*          -FP(3)     | LOG_SIZE    |
*          -FP(2)     | FFT_SIZE    |
*          -FP(1)     | return addr |
*          -FP(0)     | old FP      |
*                   +-------------+
*
*************************************************************************
****
*
*
*          WARNING: DP initialised only once in the program. Be wary
*                 with interrupt service routines. Make sure interrupt
*                 service routines save the DP pointer.
*
*          WARNING: The DEST_ADDR must be aligned such that the first
*                 LOG_SIZE bits are zero (this is not checked by the
*                 program).
*
*************************************************************************
****
*
* REGISTERS USED: R0, R1, R2, R3, R4, R5, R6, R7
*             AR0, AR1, AR2, AR3, AR4, AR5, AR6, AR7
*             IR0, IR1
```

```
*              RC, RS, RE
*              DP
*
* MEMORY REQUIREMENTS:  Program = 405 Words (approximately)
*                       Data    =   7 Words
*                       Stack =  12 Words
*
*************************************************************************
****
*
* BENCHMARKS:      Assumptions       - Program in RAM0
*                                    - Reserved data in RAM0
*                                    - Stack on Primary/Expansion Bus RAM
*                                    - Sine/Cosine tables in RAM0
*                                    - Processing and data destination in RAM1.
*                                    - Primary/Expansion Bus RAM, 0 wait state.
*
*            FFT Size    Bit Reversing     Data Source Cycles(C30)
*            --------    -------------     ----------- -----------
*                 1024             OFF              RAM1             19816
approx.
*                Note: This number does not include the C callable overheads.
*                      Add 57 cycles for these overheads.
*
*************************************************************************
****

ENBL_GIE        .set    2000h       ;global interrupt enable
FP          .set  AR3

            .global     _ffft_rl    ; Entry execution point.

FFT_SIZE:   .usect      ".fftdata",1      ; Reserve memory for arguments.
LOG_SIZE:   .usect      ".fftdata",1
SOURCE_ADDR:      .usect      ".fftdata",1
DEST_ADDR:  .usect      ".fftdata",1
SINE_TABLE: .usect      ".fftdata",1
BIT_REVERSE:      .usect      ".fftdata",1
SEPARATION: .usect      ".fftdata",1

;
; Initialise C Function.
;

            .sect ".ffttext"

_ffft_rl:   PUSH  FP          ; Preserve environment.
            LDI   SP,FP
            PUSH  R0
            PUSH  R1
            PUSH  R2
            PUSH  R3
            PUSH  R4
            PUSH  R5
                PUSH    R6
            PUSH  R7
            PUSH  AR0
            PUSH  AR1
```

```
            PUSH  AR2
            PUSH  AR3
            PUSH  AR4
            PUSH  AR5
            PUSH  AR6
            PUSH  AR7
                PUSH    IR0
                PUSH    IR1
            PUSH  DP

            LDP   FFT_SIZE    ; Initialise DP pointer.

            LDI   *-FP(2),R0  ; Move arguments from stack.
            STI   R0,@FFT_SIZE
            LDI   *-FP(3),R0
            STI   R0,@LOG_SIZE
            LDI   *-FP(4),R0
            STI   R0,@SOURCE_ADDR
            LDI   *-FP(5),R0
            STI   R0,@DEST_ADDR
            LDI   *-FP(6),R0
            STI   R0,@SINE_TABLE
            LDI   *-FP(7),R0
            STI   R0,@BIT_REVERSE

;
; Check Bit Reversing Mode (on or off).
;
; BIT_REVERSING =  0, then OFF (no bit reversing).
; BIT_REVERSING <> 0, Then ON.
;
            LDI   @BIT_REVERSE,R0
            CMPI  0,R0
            BZ    MOVE_DATA

;
; Check Bit Reversing Type.
;
; If SourceAddr =  DestAddr, Then In Place Bit Reversing.
; If SourceAddr <> DestAddr, Then Standard Bit Reversing.
;

            LDI   @SOURCE_ADDR,R0
            CMPI  @DEST_ADDR,R0
            BEQ   IN_PLACE

;
; Bit reversing Type 1 (From Source to Destination).
;
; NOTE: abs(SOURCE_ADDR - DEST_ADDR) must be > FFT_SIZE, this is not
checked.
;

            LDI   @FFT_SIZE,R0
            SUBI  2,R0
            LDI   @FFT_SIZE,IR0
            LSH   -1,IR0                  ; IRO = Half FFT size.
            LDI   @SOURCE_ADDR,AR0
```

```
            LDI   @DEST_ADDR,AR1

            LDF   *AR0++,R1

            RPTS    R0
            LDF   *AR0++,R1
           ||     STF   R1,*AR1++(IR0)B

            STF   R1,*AR1++(IR0)B

                BR      START

;
; In Place Bit Reversing.
;

            ; Bit Reversing On Even Locations, 1st Half Only.

IN_PLACE:   LDI   @FFT_SIZE,IR0
            LSH   -2,IR0            ; IRO = Quarter FFT size.
            LDI   2,IR1

            LDI   @FFT_SIZE,RC
            LSH   -2,RC
            SUBI  3,RC
            LDI   @DEST_ADDR,AR0
            LDI   AR0,AR1
            LDI   AR0,AR2


                NOP     *AR1++(IR0)B
            NOP   *AR2++(IR0)B
            LDF   *++AR0(IR1),R0
            LDF   *AR1,R1
            CMPI  AR1,AR0           ; Xchange Locations only if AR0<AR1.
            LDFGT R0,R1
            LDFGT *AR1++(IR0)B,R1

            RPTB  BITRV1
            LDF   *++AR0(IR1),R0
           || STF     R0,*AR0
            LDF     *AR1,R1
           || STF     R1,*AR2++(IR0)B
            CMPI  AR1,AR0
            LDFGT R0,R1
BITRV1:           LDFGT   *AR1++(IR0)B,R0

            STF   R0,*AR0
                  STF   R1,*AR2

            ; Perform Bit Reversing On Odd Locations, 2nd Half Only.

            LDI   @FFT_SIZE,RC
            LSH   -1,RC
            LDI   @DEST_ADDR,AR0
            ADDI  RC,AR0
            ADDI  1,AR0
```

```
            LDI   AR0,AR1
            LDI   AR0,AR2
            LSH   -1,RC
            SUBI  3,RC

            NOP   *AR1++(IR0)B
            NOP   *AR2++(IR0)B
            LDF   *++AR0(IR1),R0
            LDF   *AR1,R1
            CMPI  AR1,AR0           ; Xchange Locations only if AR0<AR1.
            LDFGT R0,R1
            LDFGT *AR1++(IR0)B,R1

            RPTB  BITRV2
            LDF   *++AR0(IR1),R0
           || STF     R0,*AR0
            LDF     *AR1,R1
           || STF     R1,*AR2++(IR0)B
            CMPI  AR1,AR0
            LDFGT R0,R1
BITRV2:           LDFGT   *AR1++(IR0)B,R0

            STF   R0,*AR0
                  STF   R1,*AR2

            ; Perform Bit Reversing On Odd Locations, 1st Half Only.

            LDI   @FFT_SIZE,RC
            LSH   -1,RC
            LDI   RC,IR0
            LDI   @DEST_ADDR,AR0
            LDI   AR0,AR1
            ADDI  1,AR0
            ADDI  IR0,AR1
            LSH   -1,RC
            LDI   RC,IR0
            SUBI  2,RC

            LDF   *AR0,R0
            LDF   *AR1,R1

            RPTB    BITRV3
            LDF   *++AR0(IR1),R0
           || STF R0,*AR1++(IR0)B
BITRV3:           LDF   *AR1,R1
           || STF R1,*-AR0(IR1)

            STF   R0,*AR1
                STF     R1,*AR0

            BR    START

;
; Check Data Source Locations.
;
; If SourceAddr =  DestAddr, Then do nothing.
; If SourceAddr <> DestAddr, Then move data.
;
```

```
MOVE_DATA:  LDI   @SOURCE_ADDR,R0
            CMPI  @DEST_ADDR,R0
            BEQ   START

            LDI   @FFT_SIZE,R0
            SUBI  2,R0
            LDI   @SOURCE_ADDR,AR0
            LDI   @DEST_ADDR,AR1

            LDF   *AR0++,R1

            RPTS    R0
            LDF   *AR0++,R1
           ||     STF   R1,*AR1++

            STF   R1,*AR1

;
; Perform first and second FFT loops.
;
;     _      _____
;    |  AR1 -> |__I1__| 0  <-  [X(I1) + X(I2)] + [X(I3) + X(I4)]
;    |  AR2 -> |__I2__| 1  <-  [X(I1) - X(I2)]
;    |  AR3 -> |__I3__| 2  <-  [X(I1) + X(I2)] - [X(I3) + X(I4)]
;    |_ AR4 -> |__I4__| 3  <- -[X(I3) - X(I4)]
;       AR1 -> |_____| 4
;              |  .  |
;              .
;                .
;               \|/
;

START:          LDI   @DEST_ADDR,AR1
            LDI   AR1,AR2
            LDI   AR1,AR3
            LDI   AR1,AR4
            ADDI  1,AR2
            ADDI  2,AR3
            ADDI    3,AR4
            LDI   4,IR0
            LDI   @FFT_SIZE,RC
            LSH   -2,RC
            SUBI  2,RC

            LDF   *AR2,R0           ; R0 = X(I2)
           || LDF     *AR3,R1             ; R1 = X(I3)
            ADDF3 R1,*AR4,R4        ; R4 = X(I3) + X(I4)
                SUBF3   R1,*AR4++(IR0),R5 ; R5 = -[X(I3) - X(I4)] --+
            SUBF3 R0,*AR1,R6        ; R6 = X(I1) - X(I2) --+  |
            ADDF3   R0,*AR1++(IR0),R7     ; R7 = X(I1) + X(I2)   |  |
              ADDF3     R7,R4,R2                ; R2 = R7 + R4 -----+  |
|
              SUBF3     R4,R7,R3          ; R3 = R7 - R4 --+  |  |  |
                                                 ;              |  |  |
|
            RPTB  LOOP1_2               ;                |  |  |  |
            LDF   *+AR2(IR0),R0         ;                |  |  |  |
           || LDF     *+AR3(IR0),R1     ;                |  |  |  |
            ADDF3 R1,*AR4,R4        ;              |  |  |  |
```

```
            || STF R3,*AR3++(IR0)          ; X(I3) <--------+  |  |  |
                 SUBF3   R1,*AR4++(IR0),R5 ;                  |  |  |
              || STF     R5,*-AR4(IR0)           ; X(I4) <-----------|--|--+
               SUBF3 R0,*AR1,R6        ;                     |  |
              || STF     R6,*AR2++(IR0)          ; X(I2) <-----------|--+
               ADDF3   R0,*AR1++(IR0),R7     ;               |
              || STF     R2,*-AR1(IR0)           ; X(I1) <-----------+
                 ADDF3     R7,R4,R2
LOOP1_2:         SUBF3   R4,R7,R3

                 STF R3,*AR3
                || STF     R5,*-AR4(IR0)
                 STF     R6,*AR2
              || STF     R2,*-AR1(IR0)


;
; Perform Third FFT Loop.
;
; Part A:
;
;     _      _____
;    |  AR1 -> |__I1__| 0  <-  X(I1) + X(I3)
;    |         |      |_____| 1
;    |         |__I2__| 2
;    |         |      |_____| 3
;    |  AR2 -> |__I3__| 4  <-  X(I1) - X(I3)
;    |         |      |_____| 5
;    |  AR3 -> |__I4__| 6  <- -X(I4)
;    |_        |      |_____| 7
;       AR1 -> |_____| 8
;              |_____| 9
;                |   .  |
;                .
;                    .
;                   \|/

               LDI   @DEST_ADDR,AR1
               LDI   AR1,AR2
               LDI   AR1,AR3
               ADDI  4,AR2
               ADDI  6,AR3
               LDI   8,IR0
               LDI   @FFT_SIZE,RC
               LSH   -3,RC
               SUBI  2,RC

               SUBF3 *AR2,*AR1,R1
               ADDF3 *AR2,*AR1,R2
               NEGF  *AR3,R3

               RPTB  LOOP3_A
               LDF   *+AR2(IR0),R0     ; R0 =  X(I3)
              || STF     R2,*AR1++(IR0)
               SUBF3 R0,*AR1,R1       ; R1 =  X(I1) - X(I3) -----+
              || STF     R1,*AR2++(IR0)  ;                        |
               ADDF3 R0,*AR1,R2  ; R2 =  X(I1) + X(I3) --+  |
              || STF     R3,*AR3++(IR0)  ;                     |  |
LOOP3_A:       NEGF  *AR3,R3           ; R3 = -X(I4) --+       |  |
```

```
                                        ;               |       |   |
          STF     R2,*AR1           ; X(I1) <-------|-------+   |
       || STF     R1,*AR2           ; X(I3) <-------|-----------+
          STF     R3,*AR3           ; X(I4) <-------+

;
; Part B:
;
;    _          _______
;   |          |_______| 0
;   |       AR0 -> |__I1__| 1 <-  X(I1) + [X(I3)*COS + X(I4)*COS]
;   |          |_______| 2
;   |       AR1 -> |__I2__| 3 <-  X(I1) - [X(I3)*COS + X(I4)*COS]
;   |          |_______| 4
;   |       AR2 -> |__I3__| 5 <- -X(I2) - [X(I3)*COS - X(I4)*COS]
;   |          |_______| 6
;   |_      AR3 -> |__I4__| 7 <-  X(I2) - [X(I3)*COS - X(I4)*COS]
;              |_______| 8
;     AR0 -> |_______| 9 NOTE: COS(2*pi/8) = SIN(2*pi/8)
;            |   .   |
;                .
;                .
;               \|/
;

          LDI   @FFT_SIZE,RC
          LSH   -3,RC
          LDI   RC,IR1
          SUBI  3,RC
          LDI   8,IR0
          LDI   @DEST_ADDR,AR0
          LDI   AR0,AR1
          LDI   AR0,AR2
          LDI   AR0,AR3
          ADDI  1,AR0
          ADDI  3,AR1
          ADDI  5,AR2
          ADDI  7,AR3

          LDI   @SINE_TABLE,AR7   ; Initialise table pointers.

          LDF   *++AR7(IR1),R7    ; R7   = COS(2*pi/8)
                                  ; *AR7 = COS(2*pi/8)

          MPYF3 *AR7,*AR2,R0      ; R0 =  X(I3)*COS

          MPYF3 *AR3,R7,R1  ; R5 =  X(I4)*COS
            ADDF3   R0,R1,R2      ; R2 =  [X(I3)*COS + X(I4)*COS]
          MPYF3 *AR7,*+AR2(IR0),R0
         || SUBF3     R0,R1,R3          ; R3 = -[X(I3)*COS - X(I4)*COS]
          SUBF3 *AR1,R3,R4  ; R4 = -X(I2) + R3 --+
          ADDF3 *AR1,R3,R4      ; R4 =  X(I2) + R3 --|--+
         || STF R4,*AR2++(IR0)    ; X(I3) <------------+  |
          SUBF3 R2,*AR0,R4         ; R4 =  X(I1) - R2 --+  |
         || STF R4,*AR3++(IR0)    ; X(I4) <------------|--+
    0     ADDF3 *AR0,R2,R4         ; R4 =  X(I1) + R2 --|--+
         || STF R4,*AR1++(IR0)  ; X(I2) <------------+  |
                                        ;                  |
          RPTB  LOOP3_B          ;                       |
```

```
            MPYF3 *AR3,R7,R1       ;                          |
           || STF R4,*AR0++(IR0)     ; X(I1) <----------------+
              ADDF3   R0,R1,R2
            MPYF3 *AR7,*+AR2(IR0),R0
           || SUBF3     R0,R1,R3
            SUBF3 *AR1,R3,R4
            ADDF3 *AR1,R3,R4
           || STF R4,*AR2++(IR0)
            SUBF3 R2,*AR0,R4
           || STF R4,*AR3++(IR0)
LOOP3_B:    ADDF3 *AR0,R2,R4
           || STF R4,*AR1++(IR0)

            MPYF3 *AR3,R7,R1
           || STF R4,*AR0++(IR0)
              ADDF3   R0,R1,R2
              SUBF3     R0,R1,R3
            SUBF3 *AR1,R3,R4
            ADDF3 *AR1,R3,R4
           || STF R4,*AR2
            SUBF3 R2,*AR0,R4
           || STF R4,*AR3
            ADDF3 *AR0,R2,R4
           || STF R4,*AR1

            STF   R4,*AR0

;
; Perform Fourth FFT Loop.
;
; Part A:
;
;     _  _____
;    |  AR1 -> |__I1__| 0  <-  X(I1) + X(I3)
;    |            |_____| 1
;    |         |_____| 2
;    |            |_____| 3
;    |         |__I2__| 4
;    |            |_____| 5
;    |         |_____| 6
;    |         |_____| 7
;    |  AR2 -> |__I3__| 8  <-  X(I1) - X(I3)
;    |            |_____| 9
;    |         |_____| 10
;    |            |_____| 11
;    |  AR3 -> |__I4__| 12 <- -X(I4)
;    |            |_____| 13
;    |         |_____| 14
;    |_           |_____| 15
;      AR1 -> |__I5__| 16
;           |_____| 17
;             |  .  |
;             .
;                .
;                \|/

            LDI   @DEST_ADDR,AR1
            LDI   AR1,AR2
            LDI   AR1,AR3
```

```
            ADDI   8,AR2
            ADDI   12,AR3
            LDI    16,IR0
            LDI    @FFT_SIZE,RC
            LSH    -4,RC
            SUBI   2,RC

            SUBF3 *AR2,*AR1,R1
            ADDF3 *AR2,*AR1,R2
            NEGF  *AR3,R3

            RPTB  LOOP4_A
            LDF   *+AR2(IR0),R0     ; R0 =  X(I3)
           || STF     R2,*AR1++(IR0)
            SUBF3 R0,*AR1,R1       ; R1 =  X(I1) - X(I3) -----+
           || STF     R1,*AR2++(IR0)  ;                        |
            ADDF3 R0,*AR1,R2  ; R2 =  X(I1) + X(I3) --+  |
           || STF     R3,*AR3++(IR0)  ;                    |   |
LOOP4_A:    NEGF  *AR3,R3           ; R3 = -X(I4) --+       |   |
                                      ;               |       |   |
              STF     R2,*AR1         ; X(I1) <-------|-------+   |
           || STF     R1,*AR2         ; X(I3) <-------|-----------+
              STF     R3,*AR3         ; X(I4) <-------+
```

```
;
; Part B:
;
;     _                 ____________
;    |                 |____________| 0
;    |       AR0        -> |__I1_(3rd)_| 1  <-  X(I1) + [X(I3)*COS + X(I4)*SIN]
;    |                   |__I1_(2nd)_| 2       .
;    |                   |__I1_(1st)_| 3       .
;    |                   |____________| 4
;    |                      |__I2_(1st)_| 5      .
;    |                   |__I2_(2nd)_| 6       .
;    |   AR1        -> |__I2_(3rd)_| 7  <-  X(I1) - [X(I3)*COS + X(I4)*SIN]
;    |                   |____________| 8
;    |       AR2        -> |__I3_(3rd)_| 9  <- -X(I2) - [X(I3)*SIN - X(I4)*COS]
;    |                   |__I3_(2nd)_| 10      .
;    |   AR4        -> |__I3_(1st)_| 11      .
;    |                   |____________| 12
;    |                   |__I4_(1st)_| 13      .
;    |                   |__I4_(2nd)_| 14      .
;    |_      AR3        -> |__I4_(3rd)_| 15 <-  X(I2) - [X(I3)*SIN - X(I4)*COS]
;                    |____________| 16
;     AR0       -> |____________| 17
;                   |     .      |
;                          .
;                          .
;                         \|/
;
```

```
            LDI   @FFT_SIZE,RC
            LSH   -4,RC
            LDI   RC,IR1
            LDI   2,IR0
            SUBI  3,RC
```

```
        LDI   @DEST_ADDR,AR0
        LDI   AR0,AR1
        LDI   AR0,AR2
        LDI   AR0,AR3
        LDI   AR0,AR4
        ADDI  1,AR0
        ADDI  7,AR1
        ADDI  9,AR2
        ADDI  15,AR3
        ADDI    11,AR4

        LDI   @SINE_TABLE,AR7
        LDF   *++AR7(IR1),R7    ; R7 = SIN(1*[2*pi/16])
                                ; *AR7 = COS(3*[2*pi/16])
        LDI   AR7,AR6
        LDF   *++AR6(IR1),R6  ; R6 = SIN(2*[2*pi/16])
                                ; *AR6 = COS(2*[2*pi/16])
        LDI   AR6,AR5
        LDF   *++AR5(IR1),R5  ; R5 = SIN(3*[2*pi/16])
                                ; *AR5 = COS(1*[2*pi/16])

        LDI   16,IR1

        MPYF3 *AR7,*AR4,R0              ; R0 =  X(I3)*COS(3)

        MPYF3 *++AR2(IR0),R5,R4 ; R4 =  X(I3)*SIN(3)
        MPYF3 *--AR3(IR0),R5,R1 ; R1 =  X(I4)*SIN(3)
        MPYF3 *AR7,*AR3,R0              ; R0 =  X(I4)*COS(3)
       || ADDF3   R0,R1,R2              ; R2 =  [X(I3)*COS + X(I4)*SIN]
        MPYF3 *AR6,*-AR4,R0
       || SUBF3     R4,R0,R3                ;  R3  =  -[X(I3)*SIN  -
X(I4)*COS]
        SUBF3 *--AR1(IR0),R3,R4 ; R4 = -X(I2) + R3 --+
        ADDF3 *AR1,R3,R4        ; R4 =  X(I2) + R3 --|--+
       || STF R4,*AR2--         ; X(I3) <------------+  |
        SUBF3 R2,*++AR0(IR0),R4         ; R4 =  X(I1) - R2 --+  |
       || STF R4,*AR3                   ; X(I4) <------------|--+
        ADDF3 *AR0,R2,R4                ; R4 =  X(I1) + R2 --|---+
       || STF R4,*AR1                   ; X(I2) <------------+   |
                                           ;                        |
        MPYF3 *++AR3,R6,R1              ;                        |
       || STF R4,*AR0                   ; X(I1) <----------------+
          ADDF3   R0,R1,R2
        MPYF3 *AR5,*-AR4(IR0),R0
       || SUBF3     R0,R1,R3
        SUBF3 *++AR1,R3,R4
        ADDF3 *AR1,R3,R4
       || STF R4,*AR2
        SUBF3 R2,*--AR0,R4
       || STF R4,*AR3
        ADDF3 *AR0,R2,R4
       || STF R4,*AR1

        MPYF3 *--AR2,R7,R4
       || STF R4,*AR0
        MPYF3 *++AR3,R7,R1
        MPYF3 *AR5,*AR3,R0
       || ADDF3   R0,R1,R2
```

```
          MPYF3   *AR7,*++AR4(IR1),R0
        || SUBF3    R4,R0,R3
         SUBF3 *++AR1,R3,R4
         ADDF3 *AR1,R3,R4
        || STF R4,*AR2++(IR1)
         SUBF3 R2,*--AR0,R4
        || STF R4,*AR3++(IR1)
         ADDF3 *AR0,R2,R4
        || STF R4,*AR1++(IR1)

         RPTB  LOOP4_B
         MPYF3 *++AR2(IR0),R5,R4
        || STF R4,*AR0++(IR1)
         MPYF3 *--AR3(IR0),R5,R1
         MPYF3 *AR7,*AR3,R0
        || ADDF3   R0,R1,R2
         MPYF3 *AR6,*-AR4,R0
        || SUBF3    R4,R0,R3
         SUBF3 *--AR1(IR0),R3,R4
         ADDF3 *AR1,R3,R4
        || STF R4,*AR2--
         SUBF3 R2,*++AR0(IR0),R4
        || STF R4,*AR3
         ADDF3 *AR0,R2,R4
        || STF R4,*AR1

         MPYF3 *++AR3,R6,R1
        || STF R4,*AR0
           ADDF3   R0,R1,R2
         MPYF3 *AR5,*-AR4(IR0),R0
        || SUBF3    R0,R1,R3
         SUBF3 *++AR1,R3,R4
         ADDF3 *AR1,R3,R4
        || STF R4,*AR2
         SUBF3 R2,*--AR0,R4
        || STF R4,*AR3
         ADDF3 *AR0,R2,R4
        || STF R4,*AR1

         MPYF3 *--AR2,R7,R4
        || STF R4,*AR0
         MPYF3 *++AR3,R7,R1
         MPYF3 *AR5,*AR3,R0
        || ADDF3   R0,R1,R2
          MPYF3   *AR7,*++AR4(IR1),R0
        || SUBF3    R4,R0,R3
         SUBF3 *++AR1,R3,R4
         ADDF3 *AR1,R3,R4
        || STF R4,*AR2++(IR1)
         SUBF3 R2,*--AR0,R4
        || STF R4,*AR3++(IR1)
LOOP4_B: ADDF3 *AR0,R2,R4
        || STF R4,*AR1++(IR1)


         MPYF3 *++AR2(IR0),R5,R4
        || STF R4,*AR0++(IR1)
         MPYF3 *--AR3(IR0),R5,R1
```

```
            MPYF3 *AR7,*AR3,R0
           || ADDF3   R0,R1,R2
            MPYF3 *AR6,*-AR4,R0
           || SUBF3     R4,R0,R3
            SUBF3 *--AR1(IR0),R3,R4
            ADDF3 *AR1,R3,R4
           || STF R4,*AR2--
            SUBF3 R2,*++AR0(IR0),R4
           || STF R4,*AR3
            ADDF3 *AR0,R2,R4
           || STF R4,*AR1

            MPYF3 *++AR3,R6,R1
           || STF R4,*AR0
              ADDF3   R0,R1,R2
            MPYF3 *AR5,*-AR4(IR0),R0
           || SUBF3     R0,R1,R3
            SUBF3 *++AR1,R3,R4
            ADDF3 *AR1,R3,R4
           || STF R4,*AR2
            SUBF3 R2,*--AR0,R4
           || STF R4,*AR3
            ADDF3 *AR0,R2,R4
           || STF R4,*AR1

            MPYF3 *--AR2,R7,R4
           || STF R4,*AR0
            MPYF3 *++AR3,R7,R1
            MPYF3 *AR5,*AR3,R0
           || ADDF3   R0,R1,R2
              SUBF3     R4,R0,R3
            SUBF3 *++AR1,R3,R4
            ADDF3 *AR1,R3,R4
           || STF R4,*AR2
            SUBF3 R2,*--AR0,R4
           || STF R4,*AR3
            ADDF3 *AR0,R2,R4
           || STF R4,*AR1

            STF   R4,*AR0

;
; Perform Remaining FFT loops (loop 4 onwards).
;
;                        LOOP
;                        1st  2nd .....
;                              \/  \/
;     |          |____X'(I1)____| 0   0  <-  X'(I1) + X'(I3)
;     |      AR1 -> |__X(I1)_(1st)_| 1    1   <-   X(I1)  +  [X(I3)*COS  +
X(I4)*SIN]
;     |          |__X(I1)_(2nd)_| 2   2       .
;     |          |__x(I1)_(3rd)_| 3   3       .
;     |          |       .      |
;     |          |       .
;     |    A -> |______________|
;     |          |____X'(I2)____| 8   16
;     |    B -> |       .      |
;     |                  .
```

```
;     |          |_____________|
;     |             |__X(I2)_(3rd)_| 13  29      .
;     |          |__X(I2)_(2nd)_| 14  30      .
;     |  AR2 -> |__X(I2)_(1st)_| 15  31 <-  X(I1) - [X(I3)*COS + X(I4)*SIN]
;     |          |____X'(I3)____| 16  32 <-  X'(I1) - X'(I3)
;     |       AR3  ->  |__X(I3)_(1st)_|  17   33  <-  -X(I2)  -  [X(I3)*SIN -
X(I4)*COS]
;     |          |__X(I3)_(2nd)_| 18  34      .
;     |          |__X(I3)_(3rd)_| 19  35      .
;     |          |       .      |
;     |                  .
;     |     C -> |_____________|
;     |          |____X'(I4)____| 24  48 <- -X'(I4)
;     |     D -> |       .      |
;     |                  .
;     |          |_____________|
;     |          |__X(I4)_(3rd)_| 29  61      .
;     |          |__X(I4)_(2nd)_| 30  62      .
;     |_      AR4  ->  |__X(I4)_(1st)_|  31   63  <-   X(I2)  -  [X(I3)*SIN -
X(I4)*COS]
;              |_____________| 32  64
;      AR1 -> |_____________| 33  65
;              |       .      |
;                       .
;                       .
;                      \|/
;

              LDI   @FFT_SIZE,IR0
              LSH   -2,IR0
              STI   IR0,@SEPARATION
              LSH   -2,IR0
              LDI   5,R5
              LDI   3,R7
              LDI   16,R6
              LDI   @DEST_ADDR,AR5
              LDI   @DEST_ADDR,AR1
              LSH   -1,IR0
              LSH   1,R7
LOOP:         ADDI  1,R7
              LSH   1,R6
              LDI   AR1,AR4
              ADDI  R7,AR1              ; AR1 points at A.
              LDI   AR1,AR2
              ADDI  2,AR2               ; AR2 points at B.
              ADDI  R6,AR4
              SUBI  R7,AR4              ; AR4 points at D.
              LDI   AR4,AR3
              SUBI  2,AR3             ; AR3 points at C.

              LDI   @SINE_TABLE,AR0 ; AR0 points at SIN/COS table.
              LDI   R7,IR1
              LDI   R7,RC

INLOP:            ADDF3   *--AR1(IR1),*++AR2(IR1),R0  ; R0 = X'(I1) + X'(I3)
--+
              SUBF3 *--AR3(IR1),*AR1++,R1       ; R1 = X'(I1) - X'(I3) -+|
              NEGF  *--AR4,R2             ; R2 = -X'(I4) --+      ||
```

```
        || STF R0,*-AR1                    ; X'(I1) <-------|------|+
         STF   R1,*AR2--                 ; X'(I3) <-------|------+
        || STF R2,*AR4++(IR1)              ; X'(I4) <-------+

         LDI   @SEPARATION,IR1   ; IR1=SEPARATION BETWEEN SIN/COS TBLS
         SUBI  3,RC

         MPYF3 *++AR0(IR0),*AR4,R4      ; R4 = X(I4)*SIN
         MPYF3 *AR0,*++AR3,R1           ; R1 = X(I3)*SIN
         MPYF3 *++AR0(IR1),*AR4,R0      ; R0 = X(I4)*COS
         MPYF3 *AR0,*AR3,R0             ; R0 = X(I3)*COS
        || SUBF3   R1,R0,R3             ; R3 = -[X(I3)*SIN - X(I4)*COS]
         MPYF3 *++AR0(IR0),*-AR4,R0
        || ADDF3     R0,R4,R2           ; R2 = X(I3)*COS + X(I4)*SIN
         SUBF3 *AR2,R3,R4         ; R4 = R3 - X(I2) --*
         ADDF3 *AR2,R3,R4         ; R4 = R3 + X(I2) --|--*
        || STF R4,*AR3++          ; X(I3) <-----------*  |
         SUBF3 R2,*AR1,R4         ; R4 = X(I1) - R2 --*  |
        || STF R4,*AR4--          ; X(I4) <-----------|--*
         ADDF3 *AR1,R2,R4         ; R4 = X(I1) + R2 --|--*
        || STF R4,*AR2--          ; X(I2) <-----------*  |
                                          ;                    |
         RPTB  IN_BLK                     ;                    |
         LDF   *-AR0(IR1),R3              ;                    |
         MPYF3 *AR4,R3,R4                 ;                    |
        || STF R4,*AR1++          ; X(I1) <--------------*
         MPYF3 *AR3,R3,R1
         MPYF3 *AR0,*AR3,R0
        || SUBF3   R1,R0,R3
         MPYF3 *++AR0(IR0),*-AR4,R0
        || ADDF3     R0,R4,R2
         SUBF3 *AR2,R3,R4
         ADDF3 *AR2,R3,R4
        || STF R4,*AR3++
         SUBF3 R2,*AR1,R4
        || STF R4,*AR4--
IN_BLK:        ADDF3 *AR1,R2,R4
        || STF R4,*AR2--

         LDF   *-AR0(IR1),R3
         MPYF3 *AR4,R3,R4
        || STF R4,*AR1++
         MPYF3 *AR3,R3,R1
         MPYF3 *AR0,*AR3,R0
        || SUBF3   R1,R0,R3
         LDI   R6,IR1
           ADDF3     R0,R4,R2
         SUBF3 *AR2,R3,R4
         ADDF3 *AR2,R3,R4
        || STF R4,*AR3++(IR1)
         SUBF3 R2,*AR1,R4
        || STF R4,*AR4++(IR1)
         ADDF3 *AR1,R2,R4
        || STF R4,*AR2++(IR1)

           STF R4,*AR1++(IR1)

         SUBI3 AR5,AR1,R0
```

```
            CMPI  @FFT_SIZE,R0
            BLTD  INLOP             ; LOOP BACK TO THE INNER LOOP
            LDI   @SINE_TABLE,AR0         ; AR0 POINTS TO SIN/COS TABLE
            LDI   R7,IR1
            LDI   R7,RC

            ADDI  1,R5
            CMPI  @LOG_SIZE,R5
            BLED  LOOP
            LDI   @DEST_ADDR,AR1
            LSH   -1,IR0
            LSH   1,R7

;
; Return to C environment.
;

            POP   DP    ; Restore  environment variables.
                POP     IR1
                POP     IR0
            POP   AR7
            POP   AR6
            POP   AR5
            POP   AR4
            POP   AR3
            POP   AR2
            POP   AR1
            POP   AR0
            POP   R7
            POP   R6
            POP   R5
            POP   R4
            POP   R3
            POP   R2
            POP   R1
            POP   R0
            POP   FP
            RETS

            .end

*
* No more.
*
*******************************************************************************
****
```

```
*******************************************************************************
*
*
*     warntwid.asm
*
*
*
*     FFT_SIZE: 512
*
*
*
*******************************************************************************
          .global       sinetab
          .data

sinetab
          .float        0.00000000000
          .float        0.01227153829
          .float        0.02454122852
          .float        0.03680722294
          .float        0.04906767433
          .float        0.06132073630
          .float        0.07356456360
          .float        0.08579731234
          .float        0.09801714033
          .float        0.11022220729
          .float        0.12241067520
          .float        0.13458070851
          .float        0.14673047446
          .float        0.15885814333
          .float        0.17096188876
          .float        0.18303988796
          .float        0.19509032202
          .float        0.20711137619
          .float        0.21910124016
          .float        0.23105810828
          .float        0.24298017990
          .float        0.25486565960
          .float        0.26671275748
          .float        0.27851968939
          .float        0.29028467725
          .float        0.30200594932
          .float        0.31368174040
          .float        0.32531029216
          .float        0.33688985339
          .float        0.34841868025
          .float        0.35989503654
          .float        0.37131719395
          .float        0.38268343236
          .float        0.39399204006
          .float        0.40524131400
          .float        0.41642956010
          .float        0.42755509343
          .float        0.43861623854
          .float        0.44961132965
          .float        0.46053871096
```

```
.float          0.47139673683
.float          0.48218377208
.float          0.49289819223
.float          0.50353838372
.float          0.51410274419
.float          0.52458968268
.float          0.53499761989
.float          0.54532498842
.float          0.55557023302
.float          0.56573181078
.float          0.57580819142
.float          0.58579785746
.float          0.59569930449
.float          0.60551104140
.float          0.61523159058
.float          0.62485948814
.float          0.63439328416
.float          0.64383154289
.float          0.65317284295
.float          0.66241577759
.float          0.67155895485
.float          0.68060099779
.float          0.68954054474
.float          0.69837624941
.float          0.70710678119
.float          0.71573082528
.float          0.72424708295
.float          0.73265427167
.float          0.74095112535
.float          0.74913639452
.float          0.75720884651
.float          0.76516726562
.float          0.77301045336
.float          0.78073722857
.float          0.78834642763
.float          0.79583690461
.float          0.80320753148
.float          0.81045719825
.float          0.81758481315
.float          0.82458930279
.float          0.83146961230
.float          0.83822470555
.float          0.84485356525
.float          0.85135519310
.float          0.85772861000
.float          0.86397285612
.float          0.87008699111
.float          0.87607009419
.float          0.88192126435
.float          0.88763962040
.float          0.89322430120
.float          0.89867446569
.float          0.90398929312
.float          0.90916798309
.float          0.91420975570
.float          0.91911385169
.float          0.92387953251
.float          0.92850608047
```

```
.float          0.93299279884
.float          0.93733901191
.float          0.94154406518
.float          0.94560732538
.float          0.94952818059
.float          0.95330604035
.float          0.95694033573
.float          0.96043051942
.float          0.96377606579
.float          0.96697647104
.float          0.97003125319
.float          0.97293995221
.float          0.97570213004
.float          0.97831737072
.float          0.98078528040
.float          0.98310548743
.float          0.98527764239
.float          0.98730141816
.float          0.98917650996
.float          0.99090263543
.float          0.99247953460
.float          0.99390697000
.float          0.99518472667
.float          0.99631261218
.float          0.99729045668
.float          0.99811811290
.float          0.99879545620
.float          0.99932238459
.float          0.99969881869
.float          0.99992470183
.float          0.99999999999
.float          0.99992470183
.float          0.99969881869
.float          0.99932238459
.float          0.99879545620
.float          0.99811811290
.float          0.99729045668
.float          0.99631261218
.float          0.99518472667
.float          0.99390697000
.float          0.99247953460
.float          0.99090263543
.float          0.98917650996
.float          0.98730141816
.float          0.98527764239
.float          0.98310548743
.float          0.98078528040
.float          0.97831737072
.float          0.97570213004
.float          0.97293995221
.float          0.97003125319
.float          0.96697647104
.float          0.96377606579
.float          0.96043051942
.float          0.95694033573
.float          0.95330604035
.float          0.94952818059
.float          0.94560732538
```

```
.float      0.94154406518
.float      0.93733901191
.float      0.93299279884
.float      0.92850608047
.float      0.92387953251
.float      0.91911385169
.float      0.91420975570
.float      0.90916798309
.float      0.90398929312
.float      0.89867446569
.float      0.89322430120
.float      0.88763962040
.float      0.88192126435
.float      0.87607009419
.float      0.87008699111
.float      0.86397285612
.float      0.85772861000
.float      0.85135519311
.float      0.84485356525
.float      0.83822470555
.float      0.83146961230
.float      0.82458930278
.float      0.81758481315
.float      0.81045719825
.float      0.80320753148
.float      0.79583690461
.float      0.78834642763
.float      0.78073722857
.float      0.77301045336
.float      0.76516726562
.float      0.75720884651
.float      0.74913639452
.float      0.74095112536
.float      0.73265427167
.float      0.72424708295
.float      0.71573082528
.float      0.70710678119
.float      0.69837624941
.float      0.68954054474
.float      0.68060099780
.float      0.67155895485
.float      0.66241577759
.float      0.65317284295
.float      0.64383154289
.float      0.63439328416
.float      0.62485948814
.float      0.61523159058
.float      0.60551104140
.float      0.59569930449
.float      0.58579785746
.float      0.57580819142
.float      0.56573181078
.float      0.55557023302
.float      0.54532498842
.float      0.53499761989
.float      0.52458968268
.float      0.51410274419
.float      0.50353838373
```

```
.float          0.49289819223
.float          0.48218377208
.float          0.47139673683
.float          0.46053871096
.float          0.44961132965
.float          0.43861623854
.float          0.42755509343
.float          0.41642956010
.float          0.40524131400
.float          0.39399204006
.float          0.38268343236
.float          0.37131719395
.float          0.35989503654
.float          0.34841868025
.float          0.33688985339
.float          0.32531029216
.float          0.31368174040
.float          0.30200594932
.float          0.29028467725
.float          0.27851968939
.float          0.26671275748
.float          0.25486565961
.float          0.24298017990
.float          0.23105810828
.float          0.21910124016
.float          0.20711137619
.float          0.19509032202
.float          0.18303988796
.float          0.17096188876
.float          0.15885814334
.float          0.14673047446
.float          0.13458070851
.float          0.12241067520
.float          0.11022220729
.float          0.09801714033
.float          0.08579731234
.float          0.07356456360
.float          0.06132073630
.float          0.04906767433
.float          0.03680722294
.float          0.02454122852
.float          0.01227153829
```

```
/*************************************************************************/
/*    File : WARNSIS.CMD
     */
/*                                                                       */
/*    TMS320C30 EVALUATION MODULE REAL FFT                               */
/*    PROGRAM LINK COMMAND FILE                                          */
/*                                                                       */
/*    Kim Dotto June 22 1994                                             */
/*                                                                       */
/*************************************************************************/

warnsis.obj
warnfft.obj
warntwid.obj
warnmath.obj
-o warnsis.out
-m warnsis.lst
-f 0x00000000


MEMORY
{
   VECS:   org = 0        len = 0x40       /* RESERVED VECTOR LOCATIONS */
   SRAM:   org = 0x40     len = 0x3FC0     /* PRIMARY BUS SRAM (16K)    */
   RAM0:   org = 0x809800 len = 0x400      /* INTERNAL RAM0(1K)         */
   RAM1a:  org = 0x809C00 len = 0x200      /* INTERNAL RAM1(1K)         */
   RAM1b:  org = 0x809E00 len = 0x200      /* INTERNAL RAM1(1K)         */
}

SECTIONS
{
   .vecs:          {} > VECS      /* RESET/INTERRUPT VECTORS          */
   .text:          {} > SRAM         /* CODE                          */
   .tmpdata:       {} > SRAM         /* temporary calculation table   */
   .fft_3sg:    {} > RAM1a        /* 2000 Hz test data                */
   .fftsrc0:    {} > RAM1a        /* 1st 512 words of FFT SOURCE DATA */
   .fftsrc1:    {} > RAM1b        /* 2nd 512 words of FFT SOURCE DATA */
   .ffttext:    {} > RAM0         /* FFT SUBROUTINE                   */
   .fftdata:    {} > RAM0         /* FFT SUBROUTINE DATA              */
   .mtext:      {} > RAM0         /* MATH SUBROUTINE                  */
   .mdata:      {} > RAM0         /* MATH SUBROUTINE DATA             */
   .bss:        {} > RAM0         /* VARIABLES                        */
   .data:       {} > RAM0         /* ASSEMBLY CODE CONSTANTS          */
   .stack:      {} > SRAM
   .fft_voc:    {} > SRAM         /* Temporary voc file storage       */
   .aicdata:    {} > SRAM         /* AIC Info                         */
}
```