college of engineering

THE UNIVERSITY OF TENNESSEE
KNOXVILLE

INSTRUCTION RESEARCH SERVICE

TOWARD EXCELLENCE IN ENGINEERING EDUCATION, RESEARCH, AND PUBLIC SERVICE!

111792

ULTRASONIC DETECTION OF CONCEALED HANDGUNS

Interim Report - Phase III/1

March 1988

ULTRASONIC DETECTION OF CONCEALED HANDGUNS

Phase III / 1: Electronics Development for a Field Test Unit

Written by:

Vig Sherrill
David Landau

Edited by:

Thomas Moriarty

University of Tennessee
310 Perkins Hall
Knoxville, TN 37996-2030

March 1988

Interim Report - Phase III / 1

MARCH 1987 - OCTOBER 1987

ABSTRACT

In this research program the feasibility of using ultrasound to detect concealed handguns has been demonstrated (Phase I) and the development of an Ultrsonic Handgun Detection System was carried out (Phase II).

In Phase III/1 the development of the digital electronics hardware and software portions of the field test unit has begun.

The particular ultrasonic technique used is called the Modal Excitation Technique (MET). This technique uses ultrasound to excite the natural modes of vibration of the target object. The set of Natural Frequencies of an object is a distinctive 'signature' of the object; this 'signature' can be used to discriminate between different types of objects.

In Phase I of this research program, the set of Natural Frequencies (the 'signature') o eight handguns and some nongun objects commonly carried about the body were experimentally determined. It was shown that the 'signature' of the handguns could be distinguished from the 'signature' of the nonguns.

In Phase II/1, the development of electronic units for transmitting and receiving airborne ultrasound relevant to handgun detection was begun and preliminary ultrasonic systems were tested.

In Phase II/2, the development of a Field Test Unit was continued, a Demonstration Unit was designed and built, and the biohazards of airborne ultrasound were investigated.

In Phase III/1, The development of digital electronics and the associated software was begun for a Field Test Unit. A Demonstration Unit utilizing the digital electronics and software developed was designed and built.

On the basis of the progress towards the development of the Field Test Unit, and the success of the Demonstration Unit, it is recommended that further work be done in a Phase III/2 effort to complete the development of the Field Test Unit and begin field testing.

PREFACE

This report was prepared by the Engineering Science and Mechanics Department, Univerity of Tennessee, 310 Perkins Hall, Knoxville, TN, 37966-2030, under Award number 83-IJ-CX-0052(S-3) for the National Institute of Justice, U.S. Department of Justice, Washington, D.C. 20531.

This Report summerizes the work done between March 1988 and October 1988, Phase III/1 - Electronics Development for a Field Test Unit.

The NIJ Program Monitor for this research program was Joseph T. Kochanski, Acting Director, CCCR.

TABLE OF CONTENTS

LIST OF FIGURES

I INTRODUCTION

1.0) Overview of previous work

Previously, in phase I and II of the NIJ project on ultrasound detection of concealed handguns, much was accomplished on the theoretical and practical sides of this problem. Primary questions, such as, do guns have patterns of modal excitation which could be used to identify them, can these modes of excitation be acoustically excited, and does the energy required for the acoustic excitement pose any biological hazard for people in the area of the dispersed energy, were asked and answered.

In phase I of this research program, the set of natural frequencies (the 'signature') of eight handguns and some nongun objects commonly carried about the body were experimentally determined. It was shown that the 'signature' of the handguns could be distinguished from the 'signature' of the nonguns.

In phase II/1, the development of electronic units for transmitting and receiving airborne ultrasound relevant to handgun detection was begun and preliminary ultrasonic systems were tested.

In phase II/2, the development of a Field Test Unit was continued, a demonstration unit was designed and built, and the biohazards of airborne ultrasound were investigated.

1.1) Overview of current work - Phase III/1

After the completion of this preliminary research, it was decided that a small, mobile, inexpensive, and quick Field Unit should be developed to carry out the ultrasonic detection of concealed handguns. In order to carry out this task, the operating philosophy was to streamline the design by using off the shelf components wherever possible to achieve the aims of reduced costs, improved reliability and availability, reduced development time, and reduced size and complexity. Also, the aim was to come up with a system which would be suitable and useful in the real world of airports, offices, and lobbies; and which would require no resident expert to operate.

The system under development for phase III of the project meets the goals of off the shelf construction and obtains the benefits which accrue from this type of design. Software was able to be written in "c" which is an excellent language for microprocessor based systems under development. The hardware / software interfaces were therefore efficiently designed and implemented. Work on the project has progressed to the point where transducers are the subsystems of major interest.

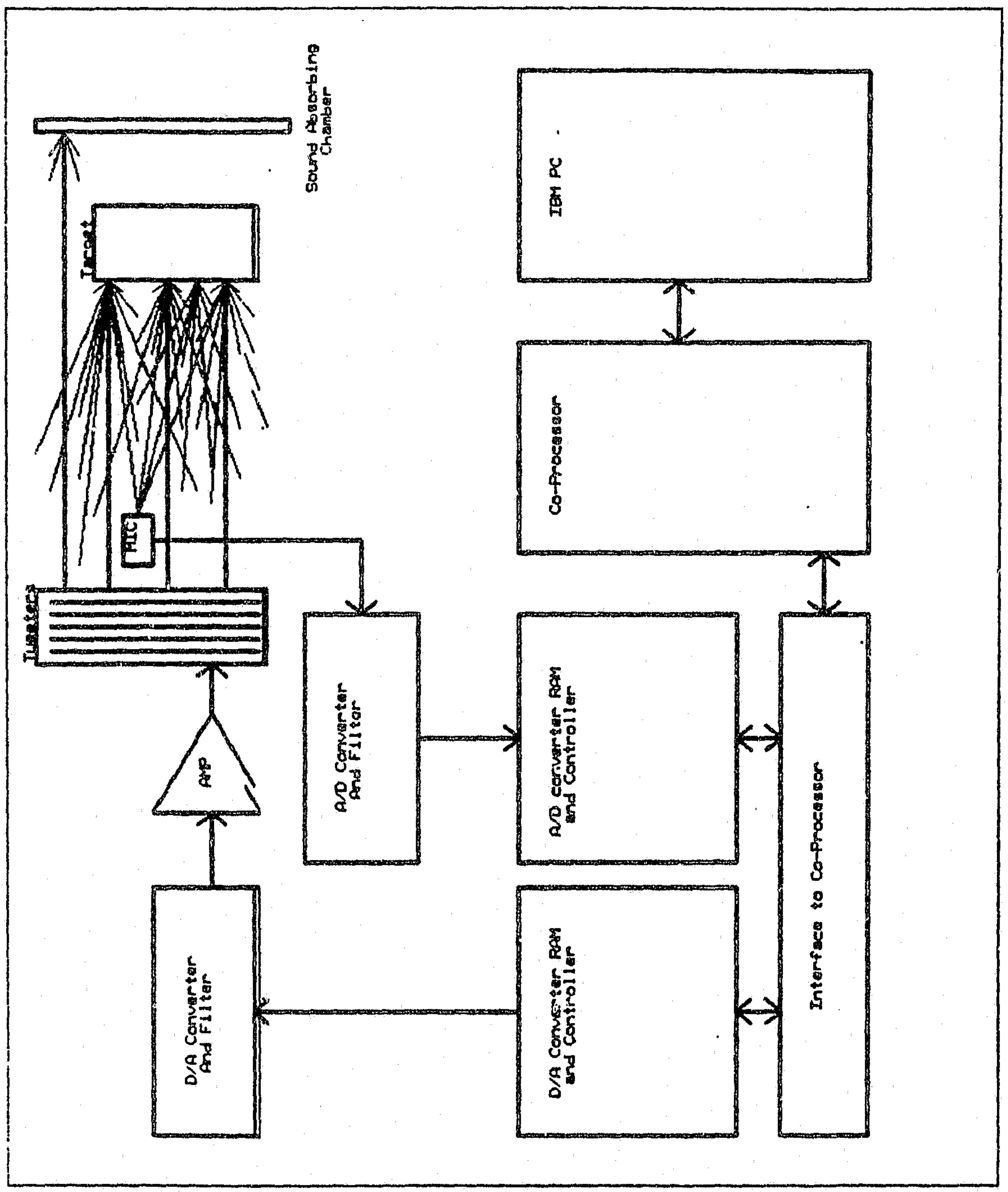
In order to achieve patterns of modal excitation which are distinguishable from the accompanying noise and reverberation in any practical Field Unit, transducers must produce sound of sufficient power and evenness to saturate a large field in which a suspect gun may be present. The gun may be concealed, in which case the sound will have to pass through the concealing material and excite the gun. The sound from the excited gun will then have to pass through the concealing material, and then a receiving transducer must pick up the resultant sound, along with whatever other sound is present, and convey it to the calculational portion of the system. Because of this sequence of events, the transmitting transducer must be able to project very loud sounds to be able to reach the gun and excite it to a significant degree through a variety of possible concealing materials. It must also be able to saturate a large field since the gun may be anywhere within this field.

On the receiving side, the transducer must be able to pick up the signal at a much diminished level from that of the transmitted signal due to attenuation through the material of concealment, and due to the dying out of the modal excitation response. The receiving transducer must also be able to pick up signals coming from anywhere within the area where a gun may be located, regardless of the direction of the gun in relation to any axis of preference for the receiver.

Because of the difficulty in satisfying these requirements for the operations of transducers in the system, this area is the probable focus of future work for a Field Unit to do ultrasonic detection of concealed handguns.

Currently, the development of the Field Unit has progressed to the point where sounds are configured to a specified design, sent, received, encoded, and analyzed in a system designed and implemented with off the shelf components.

Figure 1.0 - Block Diagram of Preliminary Field Test Unit



The rest of this report will describe the various subsections of the field unit. The report is divided into sections on the hardware comprising the unit, the software comprising the unit, and on some of the research undertaken to further develop the functionality and efficiency of the unit. Finally, a section is included on conclusions and possible future directions for the project.

II HARDWARE

2.0) Description of hardware

The system hardware and software needed to be extremely flexible because we were not sure on exactly what the final system would be like. Thus by having a very general system from the beginning we could experiment with different approaches in both hardware and software quite easily. With this in mind, the complete testing system was designed to have the following characteristics.

1) The complete system must be operational by lab technicians with little or no knowledge of the system.
2) The system needed to be somewhat portable.
3) Cost of the system should be kept minimal.
4) The system should allow for virtually any type of complex wave transmission between the frequencies of 20 and 100 Khz.
5) The system should be flexible enough that virtually any type of signal analysis could be used in signal detection.
6) The system did NOT have to be real time.

We selected early on to base our developments around the IBM PC. It provided a very large base of existing hardware and software products from which to choose from. Also because of its low cost and portability, it provided an excellent medium for upgrading towards a less flexible, faster, more efficient system.

To provide a system that would be useful in testing various different types of transited waves, and multiple signal processing algorithms, the hardware needed to be very flexible. This implies highly programmable hardware, which is another reason for choosing the IBM PC.

The general hardware system design goals are listed as follows:

1) The Transmitter must be capable of transmitting any wave that has been specified:

   a) Point by Point.
   b) By an arbitrary Formula.
   c) By an arbitrary Window.
   d) By a previously RECEIVED wave.

2) The Transmitter should have a:
   a) Fully programmable Attack time.
   b) Fully programmable Sustain time.
   c) Fully programmable Decay time.

3) The Transmitter should be accurate enough to ensure that transmitted noise would be below the background noise.

4) The Transmitter should provide at least 120 db of sound across the 20 - 100 Khz frequency spectrum.

5) The Receiver should be able to provide a minimum of 60 db of dynamic range.

6) The Receiver should be triggerable from the start of the Transmitter, with some programmable delay.

7) The Receiver frequency response should be flat to within +/- .5 db from 20 - 100 Khz.

For more detail on the usage of the system refer to the Software section.

The First approach to the hardware consisted of a board external to the IBM PC which provided the above abilities. It however was extremely slow and because of this, not very usable. To down load a typical wave would take over an hour to calculate and transfer to the board. This was deemed to slow for even a workable prototype system. A second approach was to place the system RAM inside the PC and to use a co-processor. The only external device to the PC would then be the A/D and D/A subsystem. This system worked very well and provided quite a usable system. A detailed discussion of the final hardware design follows.

2.1) Hardware Components

The hardware is comprised of several blocks connected together as shown in figure 1.0. The **Power AMP**, **Tweeters**, **Mike & Pre-amp**, are completely external to the PC where the **A/D Converter**, and **D/A Converter** have hardware internal and external to the PC. The **Co-processor Interface** and the **Co-Processor** itself are completely internal to the PC. A overview of the hardware designed follows. Because the hardware was designed making extensive usage of Programmable Logic, it is difficult to explain in detail the low level operation. In view of this, the listings for all the Programmable parts are given in the appendix along with specific comments. For more detailed schematic listings also refer to the appendix.

2.2) The Power AMP.

The Power AMP used is the Hafler series 550. This FET AMP is capable of outputting a sustained 550 watts into 4 ohms with .01% distortion from 20 to 250 khz +/- .1 db. It is also capable of outputting over 1000 watts for short periods of time under the same specs. It was chosen because of its electrical specifications, and it is commercially available for less than 500 dollars. The AMP proved to be very powerful, rugged, and extremely useful.

2.3) Tweeters.

The Transmitter chosen at first was based on the ribbon tweeters developed by Magnipan Speaker Corporation. We used these because they provided a purely resistive load of about 4 ohms, a bipolar, extremely large radiating field, and were commercially available. The first tweeters we tested with the system provide a flat frequency response up to about 60 Khz. We thought that we would be able to bring the response up to around 100khz with minimal work.

After working with the people at Magnipan, we were able to increase the frequency response up to about 78 khz (3 db point). As we suspected, the limiting factor in frequency response was the mass of the ribbon. We tried various less massive ribbons and were able to obtain even higher responses, but the ribbon became too fragile to be practical. It became apparent that this approach needed much more work to be practical. We feel that significant improvement in this area is possible.

We ended up using the transmitter array that was developed in phase II of the project. This array consists of 10 of the Polaroid ultrasonic transmitters hooked in parallel. They were mounted in a circular fashion on an aluminum plate about 6 inches in diameter. This provided a directional and small transmitter field, but we were able to transmit over 135 db of sound +/- 3 db from 40 - 100 Khz. This array is documented in detail in the last report.

2.4) Microphone and pre-amp

The Microphone & Pre-amp used were based on the designs from Phase II of the project. It used a single polaroid ultrasonic receiver with special pre-amplification and frequency correction circuitry. It proved to be very sensitive the frequencies from 35 - 100 Khz. It did not have a flat frequency response (+2 db, -8 db), but because of the digital signal processing used, were able to correct for that in software. All in all the Microphone & Pre-amp proved to very well designed, and practical. There is a detailed design of the Microphone in the last report.

We do feel that significant improvement can be made on the sensitivity of the mike. By exploiting different technologies such as a laser mike, atomic scattering, different capacitance membranes, etc. , we feel that another 10 to 20 db of sensitivity could be achieved across the frequencies of interest.

2.5) A/D Section

The A/D converter is really a stand alone hardware section that is capable of digitizing up to 128 K samples at 1 Mhz with 12 bit resolution and 11 bit accuracy. It is stand alone in that once started, it requires no intervention to operate. This was deemed necessary so that the main processor could be working on the previous samples while the A/D converter was taking new samples. In effect the A/D converter works in parallel with the main processor. The number of samples digitized is programmable from 1 sample to 128 K samples. The digitizer also contains a programmable trigger that provides delays from 1us to .128 seconds. The A/D section contains 4 main blocks as shown in figure 2.0. Because of the extreme high frequency noise in the power lines and signals, we placed the A/D converter and Filters outside the PC on a separate board, with separate power supplies. The Control section and RAM was inside the PC.

2.6) A/D Filter

The input filter section contains a Low Pass Filter (LPF) and a High Pass Filter (HPF). The LPF is used to remove the frequencies above 150 Khz. It was designed as a 6 pole Chebychev to provide a minimum of 72 db attenuation at 500 khz in order to avoid any aliasing. Its pass band ripple was +/- .3 db across the range of 20 - 100 Khz. The HPF is used to remove frequencies below 10 Khz. It was provided to remove any DC bias and noise at the lower frequencies. The circuit performed as indicated. The complete filtering circuit is shown in figure 2.1.

Figure 2.0 - A/D Converter block diagram

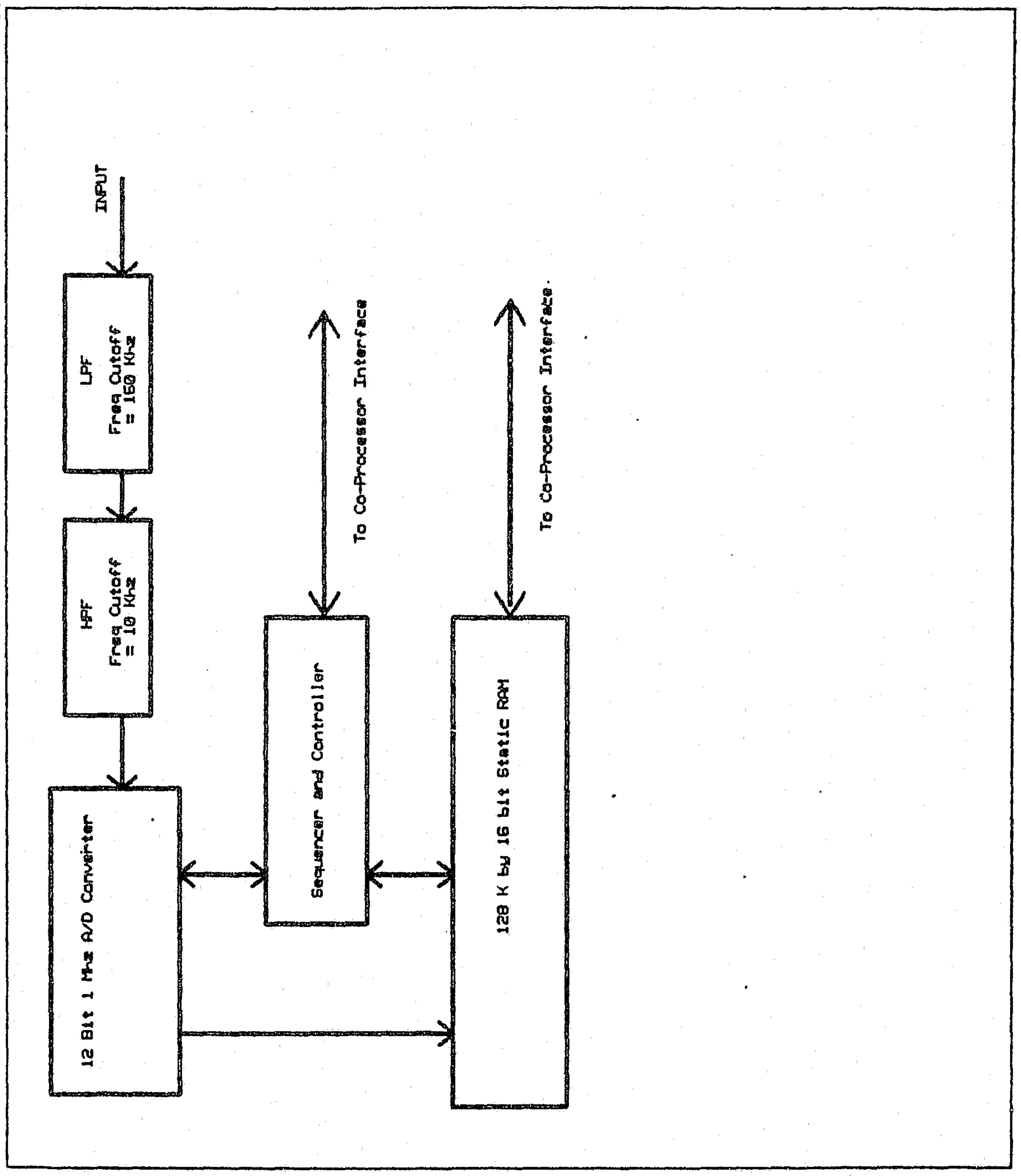
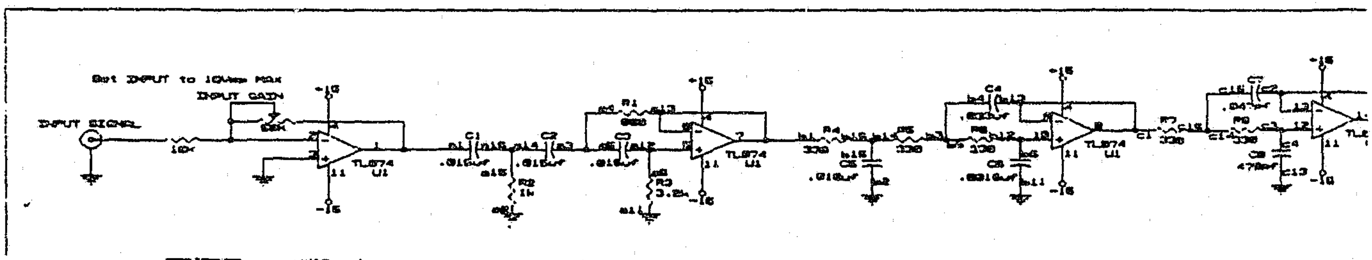


Figure 2.1 - 6 Pole Chebychev LPF for A/D Converter



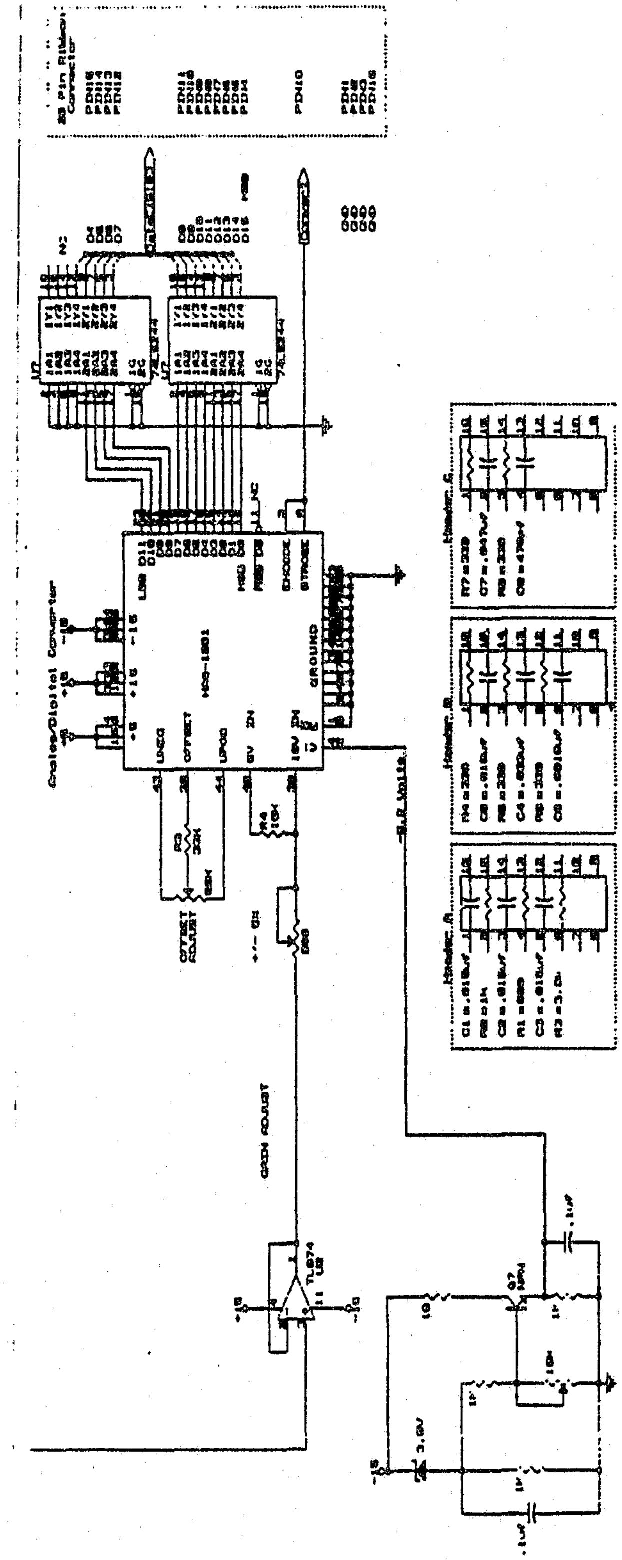
2.7) A/D Converter

The A/D converter used is an Analog Devices HAS-1201. This is a 1 Mhz 12 bit hybrid converter. We chose the high frequency conversion rate because of the extra accuracy we could get by oversampeling the input. This part proved to very reliable and accurate. We were able to achieve 11 bit accuracy and 12 bit resolution from the device. Accuracy refers to the actual digital code generated vs. the actual input voltage. Resolution refers to the total number of different digital codes possible that are monotonic. This part provided 72 db of dynamic range in theory and we were able to get about 69 db dynamic range. The loss of 3 db was due to the noise in and around the part itself. With proper circuit layout and ground planes an extra 3 db increase in dynamic range would be possible. We felt that 69 db was fine for testing, and did not pursue this. The schematic of this part is shown in Figure 2.2.

2.8) A/D Controller and RAM

The RAM is contained in the control section and would be best discussed in conjunction with the control section. The RAM for the A/D converter is made of 8, 32K by 8, Static RAMS, configured in a contiguous 128K by 16. The purpose of the RAM is to store the incoming samples in a continuous fashion. The control section is programmable with the inputs for two pointers, a control register and a trigger delay counter. The two pointers are 24 bits wide and are used as address pointers to tell the A/D converter where in RAM to START and STOP recording. This provides the

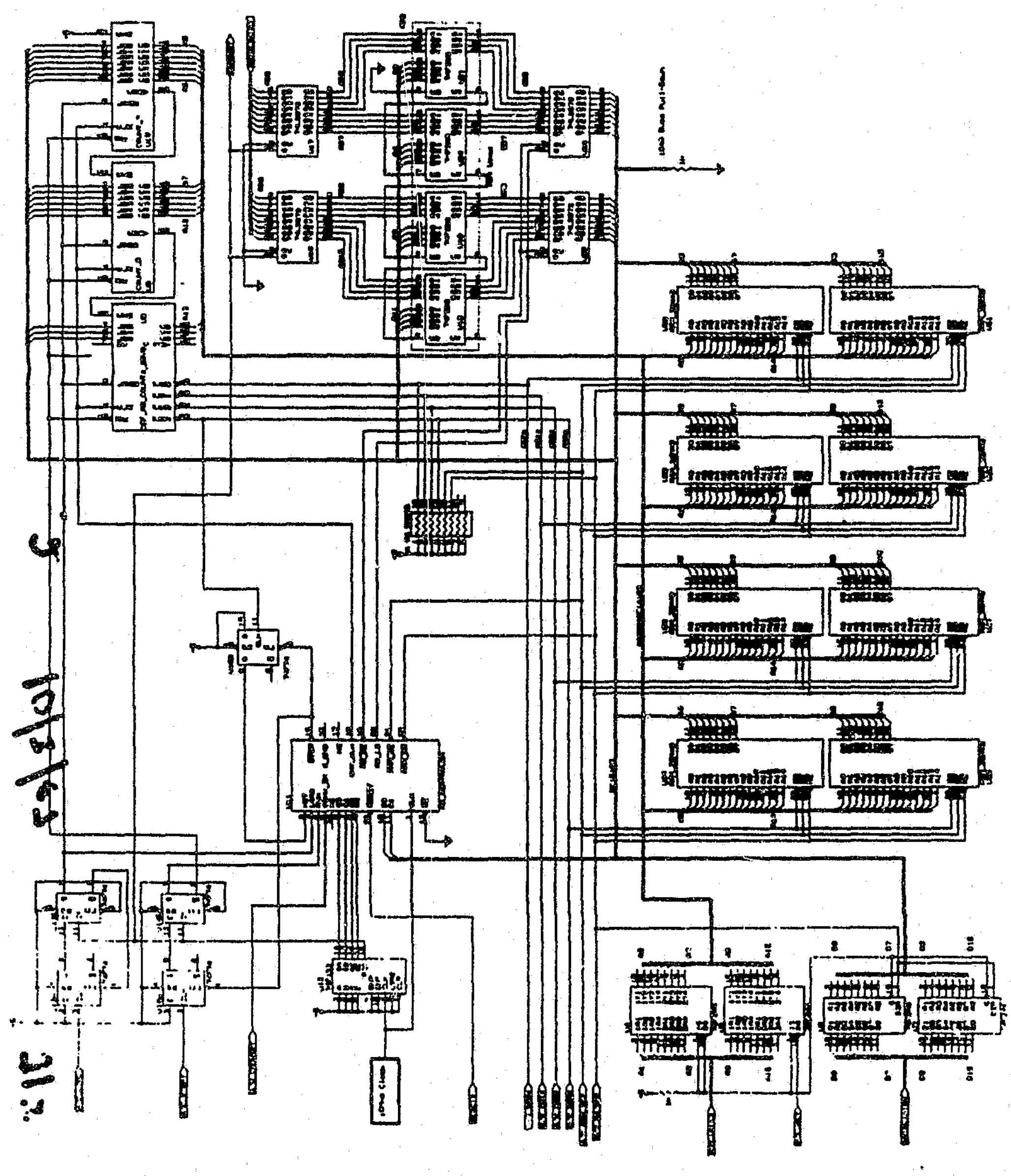
Figure 2.2 - A/D Converter IC



ability to break RAM up into separate segments for use in multiple recording. The 24 bit trigger delay counter is programmed with the number of microseconds of the delay. When started this counter counts down to 0. When 0 is reached, the A/D converter is st·rted. This counter provides the ability to delay from 1 microsecond to .128 seconds. The control register is used to control the various modes of the A/D converter section. A single bit is used to enable or disable the trigger delay. Another bit determines whether the A/D converter will WRITE over RAM when recording, or will ADD the current value from the A/D converter with what is already in RAM. This is useful for averaging. The A/D converter can ADD up to 7 samples before an overflow might occur (The RAM is 16 bits wide, 12 bits are used for any particular sample, 3 bits are allowed for overflow when adding multiple samples, and 1 bit is used in the control section).

All the logic in the control section was implemented using standard TTL and GALs (Generic Array Logic from Lattice Semiconductor). These GAL devices are electrically programmable and erasable. They were chosen because we could keep the parts count small, while allowing for a great deal of design freedom, and they are very inexpensive (each GAL can replace up to 15 or so smaller TTL devices). Because of the GALs being programmable the schematics were drawn with a different symbol for each GAL, indicating the various signals. The entire control and RAM section for the A/D converter is given in Figure 2.3.

Figure 2.3 - A/D Converter Control Diagram



2.9) D/A Section

The D/A converter is similar to the A/D converter in that it too is a stand alone system. Once started no intervention from the main processor is required. The D/A converter section is used to convert the calculated digital values into an analog form suitable for transmission. We decided to make the D/A converter a stand alone device in order to relieve the main processor so that it could be doing the spectrum analysis on the received signal. There are 4 main blocks in the D/A section as shown in Figure 2.4. The actual D/A converter and the anti-aliasing filters are external to the PC, while the D/A RAM and controller are internal to the PC.

2.10) D/A Filter

The input filter section contains a Low Pass Filter (LPF) and a High Pass Filter (HPF). The LPF is used to remove the frequencies above 150 Khz, and the HPF is used to remove any potentially damaging DC content in the system. The design, implementation, and accuracy of the filter is identical to that of the A/D filter section. The schematic for this part is shown in figure 2.5.

Figure 2.4 - D/A Converter Block Diagram

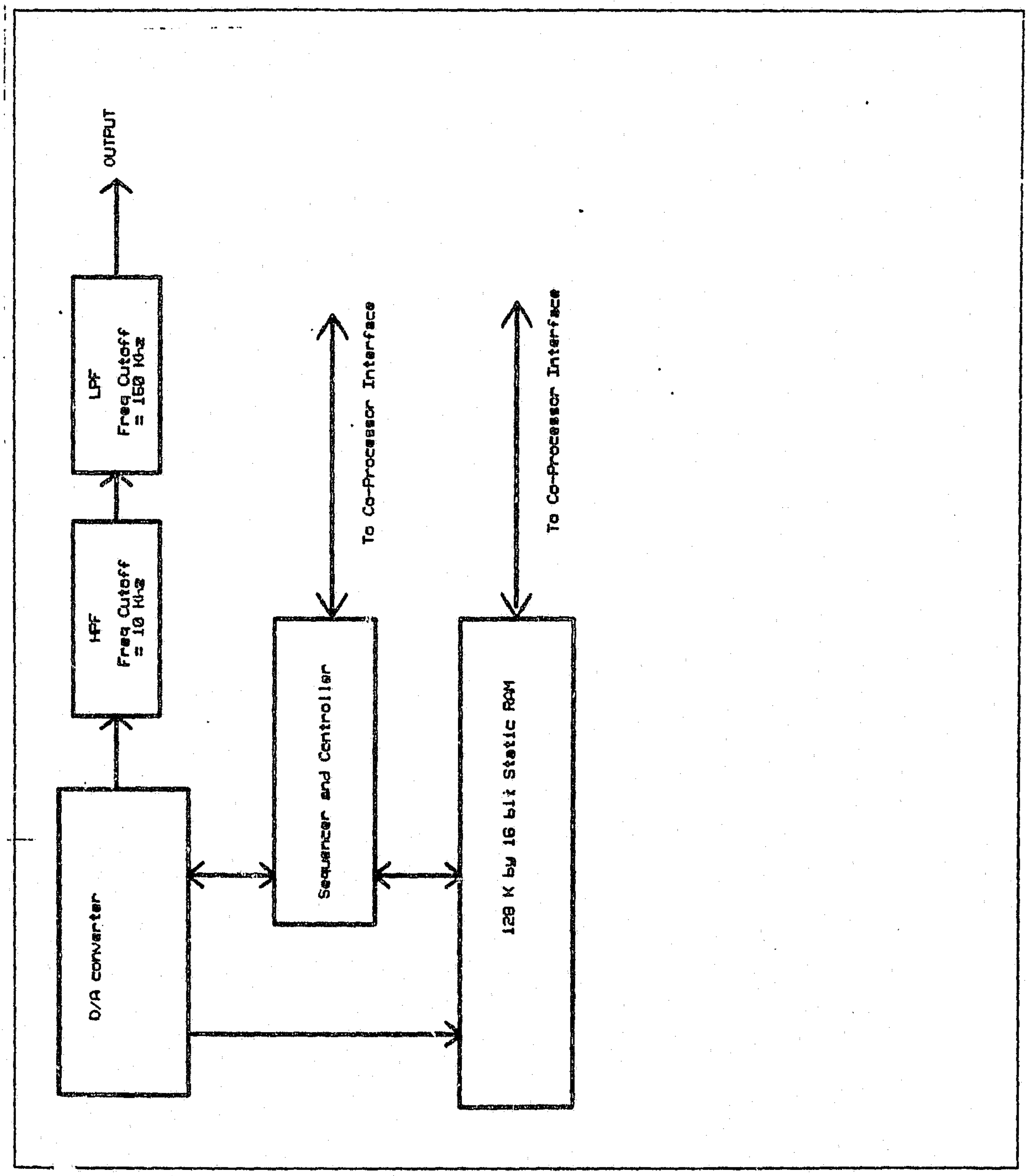
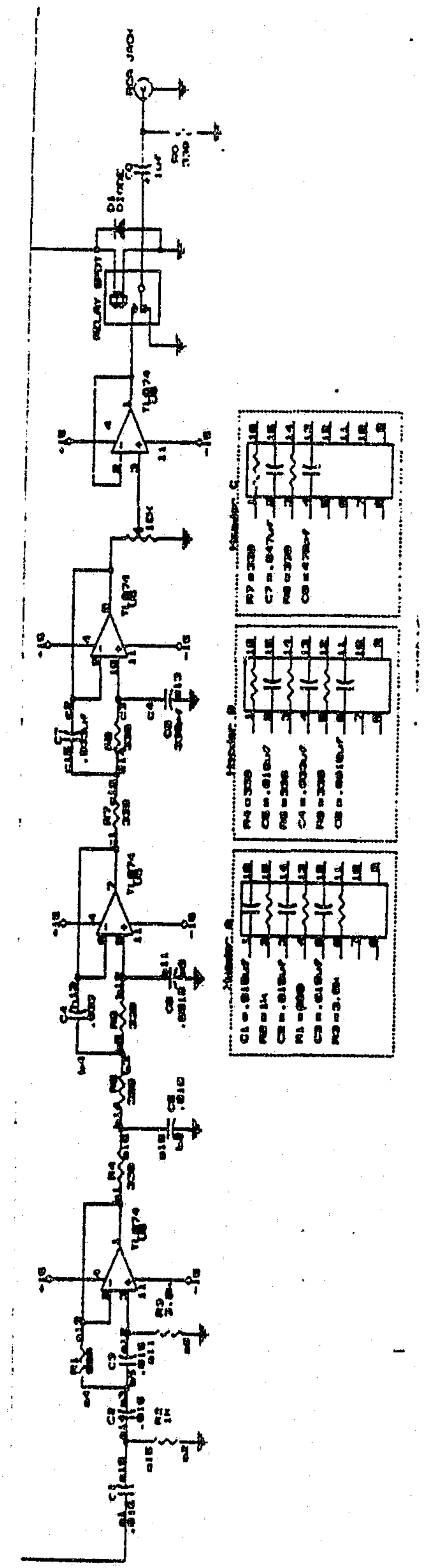


Figure 2.5 - 6 Pole Chebychev LPF for the D/A Converter



2.11) D/A Converter

The D/A converter used is a Burr Brown DAC-850-CBI-I. This is a 1 Mhz 12 bit monontonic converter. We chose the high frequency conversion rate because of the extra accuracy we get by filtering down the input to less than 120 Khz. This part was very cost effective for its accuracy. We were able to achieve better than 11 bit accuracy and 12 bit resolution from the device. We were able to get about 69 db dynamic range. [similar to the A/D converter, the 3 db loss was due mostly to the noise in and around the part itself. Again, with proper circuit layout and ground planes the extra 3 db increase in dynamic range would be possible.] After testing it became clear that we only need about 50 db of resolution, so 69 was quite adequate. The schematic of this part is shown in Figure 2.6.

2.12) D/A Controller and RAM

After the wave has been created (see software section for the actual wave creation) it is stored in D/A RAM (128K by 16 bits) by the main processor. Once stored, and the mode register set, the processor need only to issue a START command to start transmitting the wave. Each wave point is sent to the D/A converter at a rate of 1 Mhz. Once the transmission is completed the controller provides a FINISHED flag to the processor. This allows the processor to know the state of the transmitter. The mode register controls the two modes of operation of the D/A converter.

The first mode (sweep mode) is the most general but only provides up to .128 seconds of transmission. In this mode, once started, the D/A

Figure 2.6 - D/A Converter IC

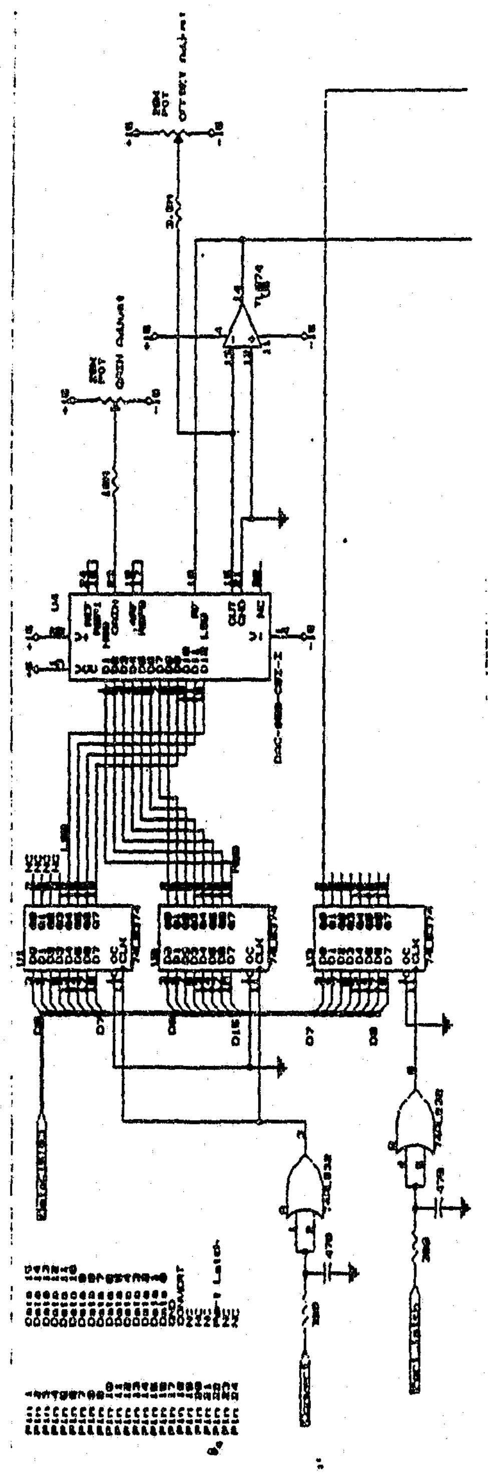
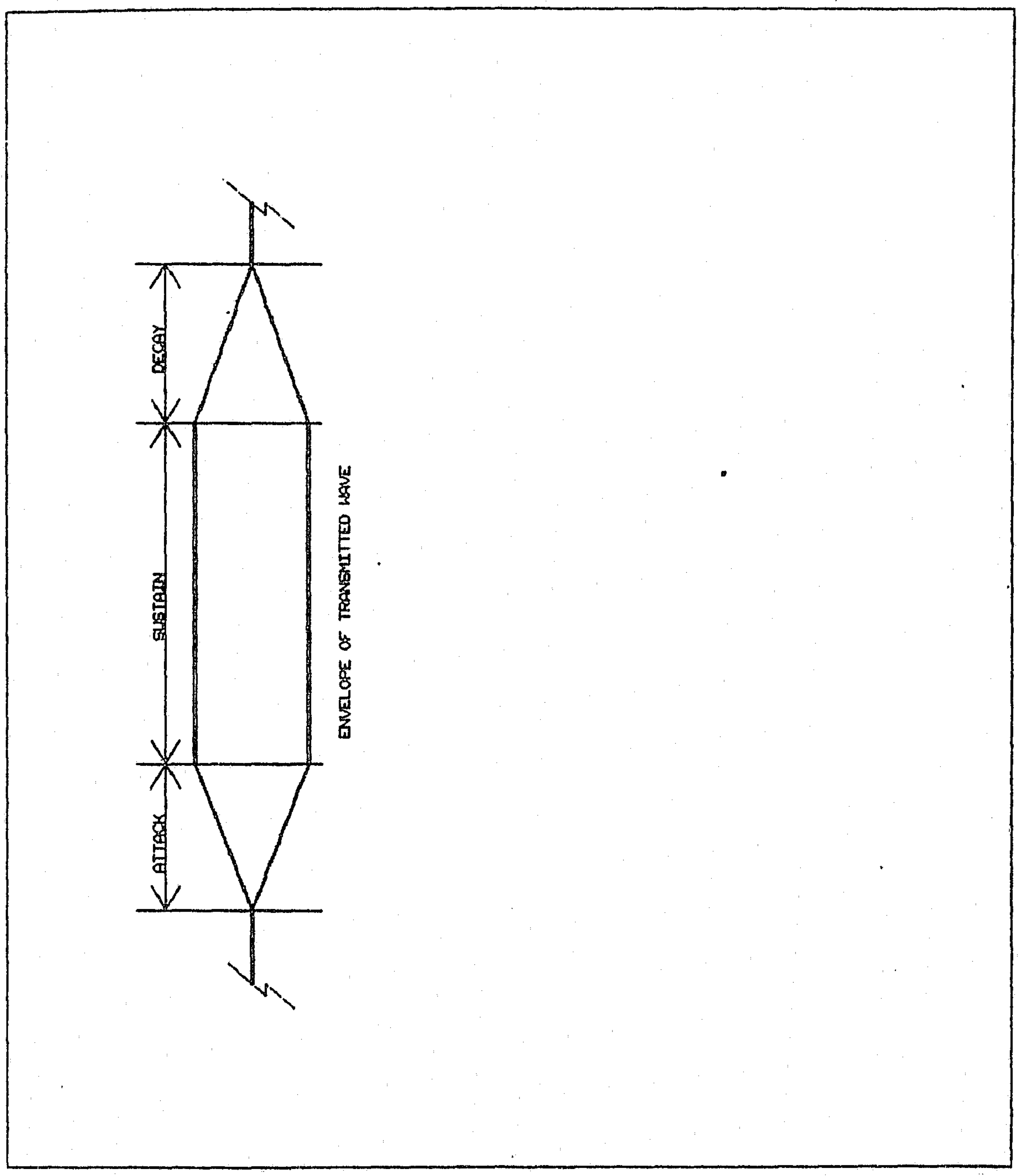


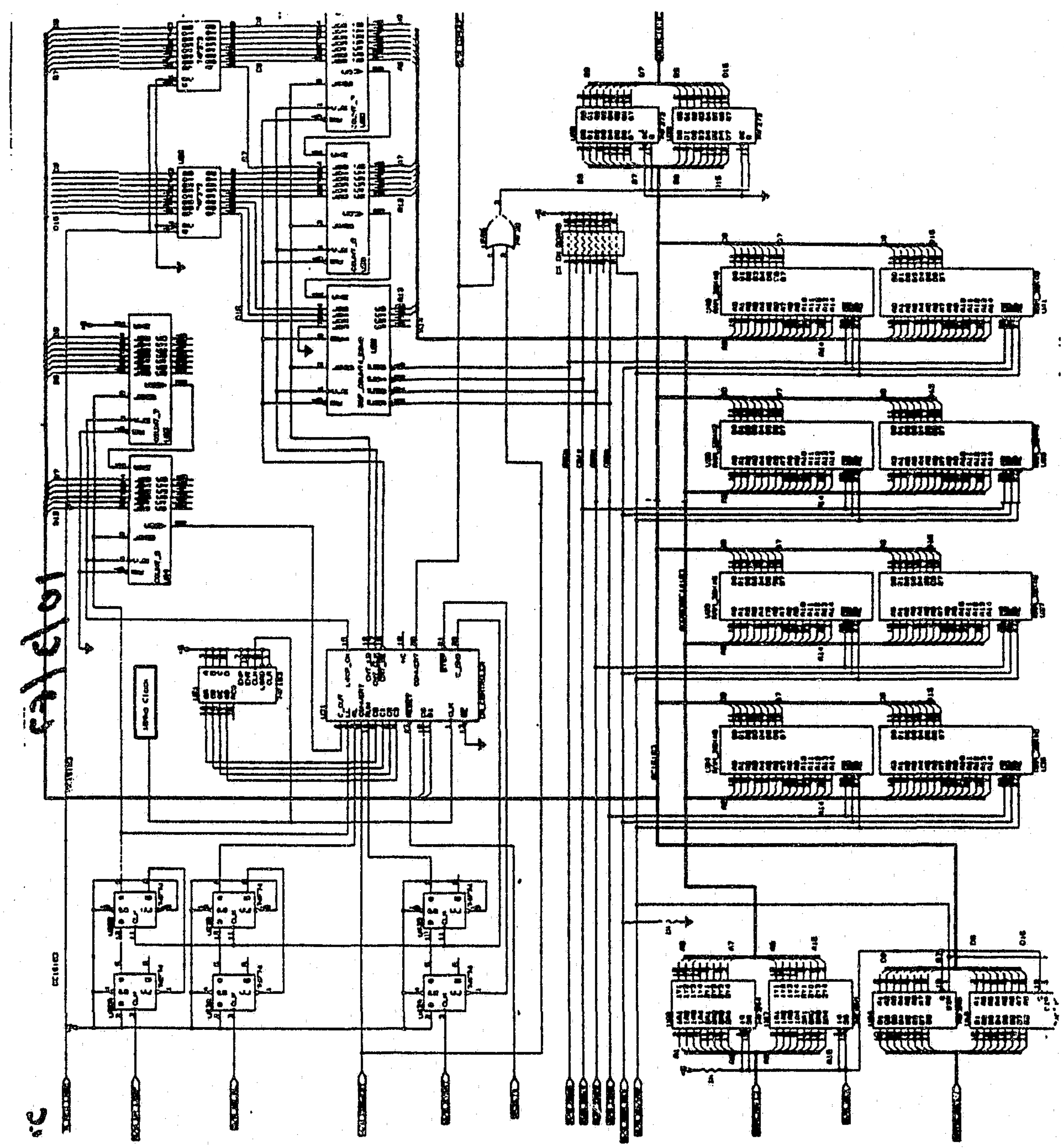
Figure 2.7 - Attack, Sustain, and Decay Waveform



section sends 12 bits of data (the lower 12) to the D/A converter starting from location 00000 in RAM and continuing sequentially until it finds a STOP bit in RAM. It does this at a rate of 1 Mhz per sample. In this mode any type of wave form is possible but only for .128 seconds.

In the second mode, (loop mode) there are three distinct sections of RAM each being an arbitrary size with the total of the three less than 128K (see figure 2.7). The first section is the ATTACK portion of the wave. This represents the initial output section of the D/A. This portion is only executed once, at the beginning of transmission. The ATTACK is used mostly for the ramp up at the beginning of transmission. If a ramp is not provided, the output is essentially windowed using a rectangular window on the data. This could provide unwanted frequencies in the transmission. By selecting different windows for transmission, different results could be obtained from the A/D converter. Also by making the ATTACK completely programmable, we could experiment with different types of windows, and determine the optimum for different conditions. The SUSTAIN section of RAM is essentially a section of RAM that when repeated will provide a completely continuous waveform. Thus at least one repeatable cycle of the waveform must fit in the SUSTAIN section. The SUSTAIN section can be repeated up to 128K times. There is a programmable register that determines the count of the sustain loop. This could provide several minutes of wave generation. The DECAY section is similar to the ATTACK section. It is used only once and is used to provide the trailing part of a window. After the SUSTAIN section is completed, the DECAY section of RAM is output to the D/A converter. The size of the three sections is completely programmable, but must not overlap. This mode proved very useful in experimenting with different transmission windows.

Figure 2.8 - D/A Converter Control Diagram



where the first mode proved very useful in transmitting a sweep of frequencies. The entire control and RAM section is shown in figure 2.8.

Again, the D/A controller section makes extensive usage of GALs. This provides for a very compact system, as well as being very easy to modify. The entire listings for GALs is given in appendix A.

2.13) Co-Processor Interface

The Co-Processor to the PC was a single internal card with a 68020 and it's own 68881 Math Co-Processor. The 68020 can provide up to 16 Gigabyte of RAM interface, however we only used an external 512K bytes. The A/D and D/A section each have 128K of 16 bit wide RAM. That RAM is also mapped into the address space of the 68020. This means that the 68020 has direct access to all the data for the system, thus completely removing any type of data transfer from one machine to another. The interface between the A/D and D/A sections was merely a method of dual porting the RAM. The only constraint is that the 68020 cannot access the memory at the same exact time as the A/D or D/A is accessing memory. Special timing circuitry was provided to prevent this from happening. The overall result was a very efficient method of communication between the Co-Processor and its A/D / D/A interfaces. The schematics for the Definicon interface are shown in Figures 2.9 and 2.10.

Figure 2.9 - Definicon Interface I

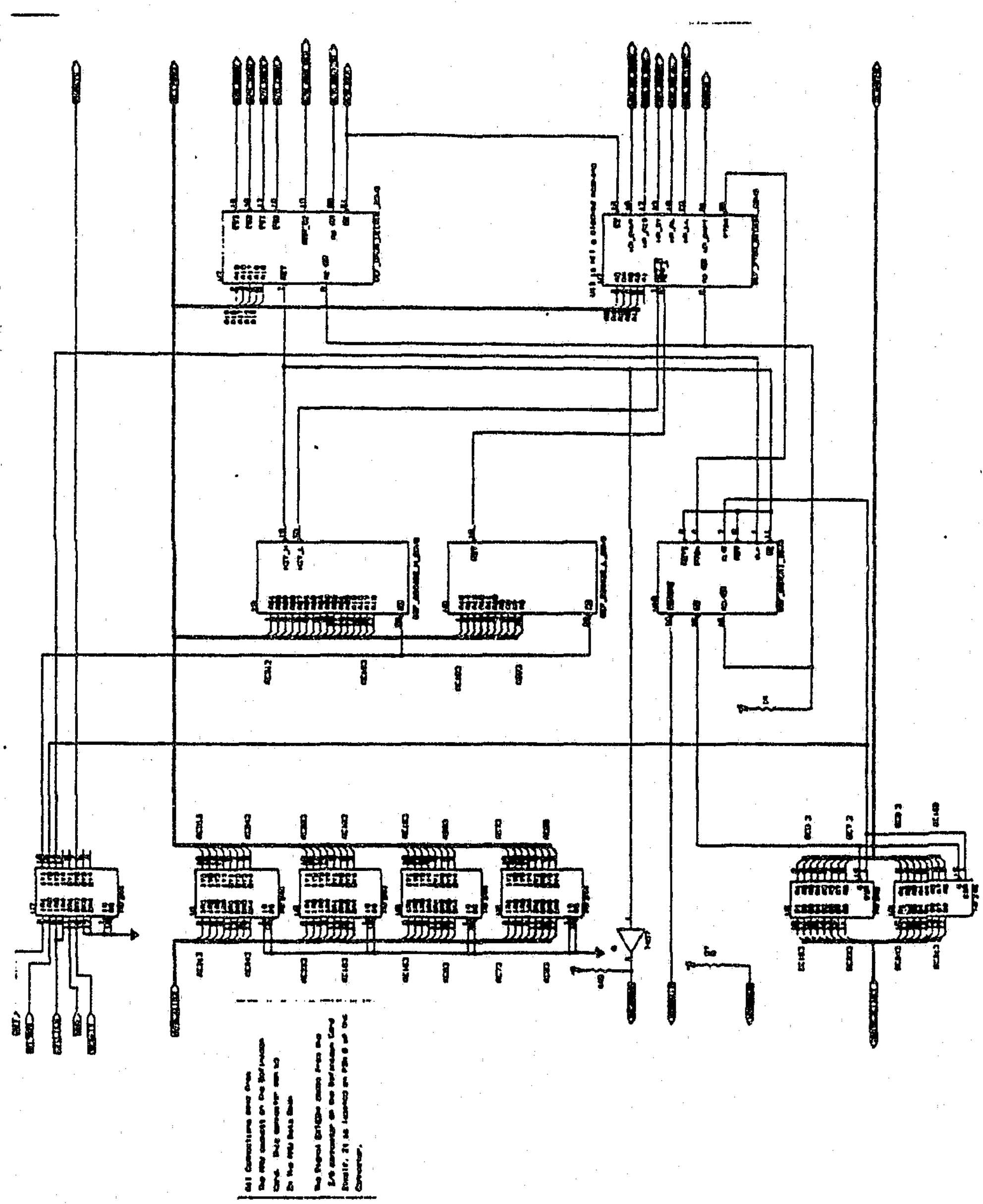
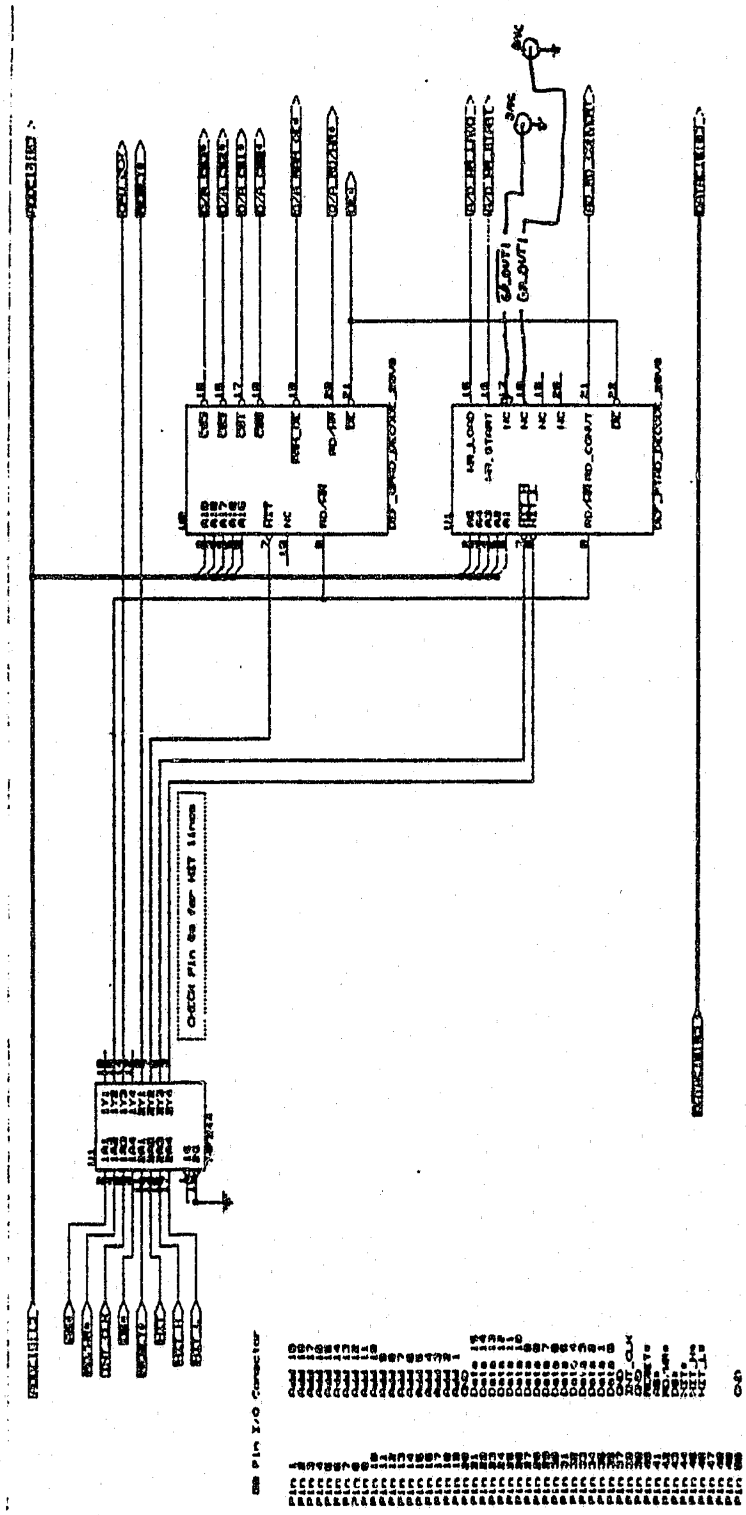


Figure 2.10 - Definicon Interface II



2.14) Co-Processor

When we first started the project we did not use a Co-Processor, but used the IBM PC as the main processor. Because of the complexity of the algorithms, this turned out to be unbearably slow (taking up to an hour to calculate the different waves for transmission alone). So we decided to add a Co-Processor to speed up the calculations. We evaluated several different co-processors and chose the DSI-780+ by Definicon Systems Inc. We chose this particular product because of it's cost/performance ratio. In raw computing power, its speed rivals that of a VAX 11-780 (thus the name DSI-780+). The entire card was purchased for less than the cost of an IBM PC-AT and provided better than 100 times the performance of the IBM PC. The Co-Processor is based upon the Motorola 68020 running at 16.5 Mhz coupled with a 68881 math processor. It contains on board 1 Meg of RAM and all the necessary hardware to interface with the PC. All of the software processing runs on the DSI-780+ while the disk I/O and display run off the IBM PC. This configuration provides a very computationaly powerful and efficient engine to run the algorithms on.

2.15) Concluding thoughts on hardware development

The final system design proved to be very efficient in space, and power consumption, as well as being extremely flexible. The hardware was fully operational as described in this report, and provided a very usable system for testing. Because of the nature of research, we wanted a system that would be very flexible to accommodate various unknowns in experimentation. However, if this were to become a commercially available product, most if not all of the hardware must be completely redesigned (the flexibility is not needed in a final unit). We determined that it was more advantageous at this stage in the project to have a usable system that could accommodate the various unknowns in hardware, and software.

The software ended up using a radix four FFT algorithm to decompose the time domain into the frequency domain. This FFT was done completely in software and (although highly optimized) would take several seconds to complete (depending on the number of points taken). Because the FFT will probably be needed for use in the final pattern recognition algorithm, it will have to be implemented in hardware. In fact, there exists several different IC chip sets available on the market to do just this. We feel that a speed increase of over 1000 would be possible using this approach.

Once a fairly good algorithm is determined, the hardware could be optimized to execute the algorithm very efficiently. This could even evolve the design of ASIC (Application Specific Integrated Circuits) to make a very small and portable system.

The completed and operational system provided a very powerful vehicle for testing the various software algorithms as well as testing different hardware sections.

III SOFTWARE

3.0) Reasons for Development scheme

A software package was needed which could perform a varying group of tasks in a flexible and easily accessible manner, growing as our development of hardware progressed. The reason for these needs has to do with the nature of hardware development in a project such as ours where differing versions of hardware, or even modifications to hardware, are frequently assembled, tested, and used. Though we knew at the outset that sound would have to be transmitted and received, and that data would have to be run through an FFT algorithm and analysed, we did not know precisely how we wanted to do these actions in the context of an overall operating package. We had to develop the package.

So, as far as software was concerned, our first need was to create a system which would drive functions, or groups of functions, from a multilayered menu driven keyboard input package, to control any piece of hardware under development. In the system which we created, the functions which drive the developing hardware can be placed in any sequence, or aggregate, in order to associate a particular logical function (such as calculating a compound wave and loading it in a section of RAM in a state prepared for transmission) with a single keystroke in a meaningful way. The adoption of this flexible approach to system design made it possible for us to quickly configure a hardware/software package allowing for the development of a field test unit for the ultrasound detection of handguns.

In order to have an automatic detector of handguns, several steps have to be taken care of. We needed to be able to transmit sounds of our own design, designed to excite the particular resonances of the particular handguns. We needed to be able to receive the sounds generated by the excited guns in a digitized form. We needed to be able to transform that data from the time domain into the frequency domain in order to analyze the frequency structure of the resonated sound. Finally, the frequency structure of the elicited sounds would have to be analyzed for patterns in order to determine what type, if any, of gun has been excited. Each of these steps requires the ability to easily get at and control the hardware. In addition, each of these steps requires the working of other smaller subsystems. For instance, to manipulate received sounds in the digital domain we needed an A to D section to be working and under control. Therefore, we needed some software to enable the ADC to be tested during it's development and use. The other functions similarly required testing and development, and therefore the need for the flexibility and power of the software system. For these reasons the software package for frequency generation and analysis was developed along the following lines:

3.1) Actual Development

At the top of the structure is a main menu. This menu allows the selection of which area of operation is to be undertaken.

The choices are:

transmission parameter selection

reception parameter selection

board control

data transfer

DAC testing

ADC testing

The actual operations for these areas are described below.

3.2) Transmission

A transmission parameter selection menu has been implemented allowing the setting of all parameters which have to do with the transmission of data. There are two basic modes of transmission which are supported. The first mode is the swept frequency mode in which frequencies are generated and are then able to be transmitted, from low to high frequency, with the incremental increase in frequency following a linear function in the time domain. The second mode used is a discrete frequency mode which allows the selection of particular discrete frequencies to be generated which are then able to be transmitted simultaneously.

Swept frequency mode - The user selects the starting and ending frequencies in the sweep specifying a range between 30 Khz and 100 Khz. These endpoint frequencies are easily alterable but were chosen for pr ical reasons. The 30 Khz starting boundary point was chosen so that no audible frequencies would be emitted at loud volumes possibly imparting auditory nerve damage to the researchers. The ending boundary point was empirically chosen to be 100 Khz because that was close to the point where our transducers ceased to be effective at transmission of desirable power levels, in addition to the fact that as the frequencies get higher they are more easily absorbed.

The user selects burst time, ramp time, and decay time in order to specify the envelope containing the frequency sweep. The ramp time is the amount of time to execute a linear ramp of power from zero to full power. The decay time is the amount of time to execute a linear decay from full power to zero. These times are variable from 0.001 msec to 10.0 msecs. The burst time is the time for the complete sweep to take place, from the beginning of the ramp at zero power to the end of the decay at zero power. The maximum burst time for the sweep mode is 131.0 msecs. The entire burst is not at full power because of the ramp up and the decay. However, the frequency range selected is transmitted at full power. An additional 5Khz is tacked on to the end of the selected frequency range and is also transmitted at full power in order to facilitate the time delay spectrometry research. First a starting frequency to an ending frequency are determined by:

```
c = tt - (rt + dt)
fpt = (fpef + 5.0 - fpsf) / c
sf = fpsf - (fpt * rt)
ef = fpef + 5.0 + (fpt * dt)
```

where

c - time of the full power region

tt - total, or burst, time

rt - ramp time

dt - decay time

fpt - frequency per time

fpef - full power ending frequency (user specified upper endpoint)

fpsf - full power starting frequency (specified lower endpoint)

sf - starting frequency

ef - ending frequency

Next, the wave is calculated to be a linearly increasing (in frequency) sinusoidal sweep from the determined starting frequency to the determined ending frequency. The wave is then scaled to reach full power. The ramp and decay functions are applied to the scaled version of the wave, and then the full wave is loaded into the transmission board's RAM, awaiting the command to begin transmission.

Wave Envelope Diagram

```
    ,---------burst time ---- -,
    ¦                          ¦
      /¦              ¦     ¦\
     / ¦              ¦     ¦ \
    <  ¦  specified   ¦5khz¦  >
 ,->\  ¦      range   ¦     ¦ / <-,
 ¦    \¦              ¦     ¦/     ¦
 ¦ramp up                          ¦decay
```

The specified frequency range is transmitted at FULL power. There is 5Khz extra at full power tacked on for time delay spectrometry investigations. The ramp up and ramp down sections, while conforming to the smooth frequency sweep, are not at full power, but are enveloped as shown.

Figure 3.0 - Envelope of sweep mode transmisson

Discrete frequency mode - The user selects the frequencies desired for simultaneous transmission, specifying any in the range between 30 Khz and 100 Khz. The maximum number of simultaneous frequencies allowed in the current package is 100. The maximum resolution is 0.1 Khz in any individual frequency. The user makes frequency selections by entering the frequency editor package which allows the selection and editing of a list of frequencies to be included in a wave. When done specifying frequencies, the user exits the editor package and is then able to choose envelope parameters.

The user selects burst time, ramp time, and decay time in order to specify the envelope containing the compound wave. The ramp time is the amount of time to execute a linear ramp of power from zero to full power. The decay time is the amount of time to execute a linear decay from full power to zero. These times are variable from 0.001 msec to 10.0 msecs. The burst time is the time for the complete wave to be transmitted, from the beginning of the ramp at zero power to the end of the decay at zero power. The maximum burst time for the discrete mode is 40000.0 msecs. A looping method is used to repeat compatible portions of the generated wave to achieve this long time of transmission.

First, the wave is calculated to be a linear combination of the sinusoids from each determined frequency. The wave is then scaled to reach full power. The ramp and decay functions are applied to the scaled version of the wave, and then the full wave is loaded into the transmission board's RAM; awaiting the command to begin transmission. Loop bits are set to cause a 10.0 msec constant portion of the wave to repeat the proper number of times to obtain the desired burst time. The 10.0 msec repeat portion "joins up" at beginning and end because of a resolution of 0.1 Khz for each designated frequency.

```
          Wave Envelope Diagram

       ,--------burst time ------,
       |                         |
          /|                   |\
         / |                   | \
       <   |                   |  >
     ,->\  |                   | / <-,
     |    \|_                  |/    |
     |ramp up                        |decay
```

Figure 3.1 - Envelope of discrete mode transmission

3.3) Reception

A reception parameter selection menu has been implemented which allows the user to control the process of receiving data through the Analog to Digital Converter. This menu allows the setting of the record time, and of the delay time.

The record time may be set anywhere within the range of 0.0 msecs to 131.0 msecs, and specifies the total time for which to gather data through the ADC and into the reception RAM. The data is loaded starting at the 0th location and extends one address every microsecond for the duration of the acquisition.

The delay time may be set anywhere within the range of 0.0 msecs to 131.0 msecs, and specifies the total time to delay the start of acquisition of data through the ADC after the transmission of data through the DAC has started.

3.4) Board control

The board control menu allows the user to control the special purpose hardware in order to carry out the actions of transmission and reception of data. The options possible from this menu are transmit, receive, transmit and receive, clear acquired data area, and transfer acquired data to transmission area.

The receive option, when invoked, causes the reception apparatus to become active, and the acquisition of data begins after the previously specified delay time has elapsed. The reception takes place for the amount of time previously specified as the reception time with the acquired data being placed in the A to D's RAM.

The transmit option, when invoked, starts the process of transmission. The data which is present in the D to A's RAM is sequentially sent out of the D to A, one value every microsecond, unless a stop bit or a loop bit is encountered in the data. The presence of a loop bit causes the loop count to be decremented and, if the loop count is not the critical value, for the current address of transmission to be changed to 10000 addresses, or 10 milliseconds, earlier. If the loop count is the critical value, or if no loop bit is encountered, the current address is merely incremented and transmission continues. Transmission ceases when a stop bit is discovered.

The transmit/receive option causes both actions to occur and for the operations described immediately above to begin at the same time. This is the usual action taken when testing an object for excitation and analysis.

The clear acquired data option, when chosen, causes the A to D ram to be zeroed. Any data acquired solo, or during averaging procedures, is cleared for new acquisitions.

The acquired data transferred to the transmission area option, when chosen, gets the data from the A to D's RAM and moves it to the D to A's RAM for transmission. Because the acquired data is 16 bits per piece of data while transmitted data goes through a 12 bit D to A converter, only the top twelve bits of the acquired data are loaded for retransmission. The bottom four bits of the RAM, which are control bits, are zeroed except for the last piece of data in which the stop bit is set.

3.5) Data transfer

The data transfer menu allows the user to control the flow of data from disk into the hardware, from the hardware to disk, and from the hardware to the screen for display. Various manipulations of the data are also allowed on the data when the display mode is current.

The current parameter set sent to file option, when chosen, causes the complete set of parameters for a current run to be sent to a disk file, along with any waveform which has been generated.

The current parameter set loaded from file option, when chosen, causes the complete set of parameters for a current run to be loaded from a disk file, along with any waveform which has been stored.

The acquired data sent to file option causes the data which has been accumulated in the A to D section of RAM during analysis to be sent to a specified file on disk.

The transfer acquired data to screen and allow for manipulation of data option opens up a special sub-package which gives access to several tools of analysis and graphical manipulation. The options supported in this section are as follows:

A starting time may be specified which is the point value, in either an array A or B into which data under question will have been moved, at which to begin a plot of the data. The time is related to the point value by the ratio of one microsecond per point, since each sample creates one point and the sample rate is 1 Mhz.

A stopping point may be specified which is the point value at which to end plotting. The endpoint applies whether the A or B array is actually plotted. A starting time (point) of 0 and a stopping time (point) of 999 would specify a plot of the first millisecond of data. A starting time of 1000 and a stopping time of 1999 would specify the use of the second millisecond of data in a plot.

A lower Y value may be set which specifies what the value is to represent the bottom of the axis in any plot. The value is a scaling factor and is used as a parameter for the plot routines. This value is only used when the autoscale mode is not currently in effect, since autoscale automatically figures the parameters for axis scaling.

An upper Y value may be set which specifies what the value is to represent the top of the axis in any plot. The value is a scaling factor and is used as a parameter for the plot routines. This value is not used when autoscaling is in effect.

A plot axis menu slot, when invoked, causes a plot axis routine to use the starting time, ending time, upper Y value, and lower Y value to create a scaled axis plot when not in autoscale mode. When autoscale mode is current, the plot of the axis uses the user specified starting and ending times, but not the upper and lower Y values. Instead the plot routine uses the data which is to be plotted to scale the Y axis.

The get data in A menu option causes the acquired data in the A to D RAM to be moved in the amount of the recorded time or 50000 microseconds, whichever is less, to the array A for subsequent manipulation or analysis.

The get data in B menu option causes the acquired data in the A to D RAM to be moved in the amount of the recorded time or 50000 microseconds, whichever is less, to the array B for subsequent manipulation or analysis.

The plot data in A menu option plots the data in array A according to the parameters of starting time, stopping time, upper Y value, lower Y value, and plot mode.

The plot data in B menu option plots the data in array B according to the parameters of starting time, stopping time, upper Y value, lower Y value, and plot mode.

The Auto FFT menu selection allows the repeated transmission of a desired waveform, and the acquisition of the resulting data created by the excited object, followed by the formation and display of the frequency information found in the specified region of that acquired data. The plot from each iteration of this cycle is in a different color.

First, the starting color is entered as a value from 0 to 15. Next, the starting point from array A at which to begin the FFT is entered. Then the exponent for the Radix four FFT is asked for which specifies the power to raise four to in order to obtain the number of points which are to be analysed. The maximum value supported for the power of radix four by this package is 7, which uses 16k points.

The axis is then plotted in autoscale mode. The data in the D to A's RAM is transmitted. The A to D section acquires data in it's RAM according to the current parameter set. The array A is filled with the data which has been acquired. The radix four FFT is performed on the specified region of the data in array A. The results of the FFT are put in array B, and are then plotted. This series of events is repeated over and over at the user's direction, creating new plots over the old of the current frequency response elicited, until the user exits from this section of the package.

The clear screen option clears the screen of all contents and plots and then provides the menu with current parameters listed.

The transmit/receive option causes the transmission of the data in the D to A's RAM. Simultaneously, the A to D section acquires data in it's RAM according to the current parameter set.

The plot mode option allows the user to specify whether the plotting is to be in autoscale mode, which means that the scaling will be automatically set to the data which is to be plotted, making the maximum and minimum values in the data correspond to the top and bottom boundaries of the plotted axis, respectively. The plot mode may also be set to absolute scaling, which means that the upper and lower values for Y, which are specified by the user in this mode, are to be used as the determining factors in the top and bottom values for the axis boundaries.

The FFT on data in A with results to B menu selection causes the following chain of actions to occur. The starting point from array A at which to begin the FFT is entered. Then the exponent for the Radix four FFT is asked for, which specifies the power to raise four to in order to obtain the number of points which are to be analysed. The maximum value supported for the power of radix four by this package is 7, which uses 16k points. Then the desired FFT is performed on the specified data and the results are put in array B.

The exit option is the last option in the screen data manipulation portion and gets the user out of this portion of code and back up to the next highest level, the data transfer menu.

3.6) DAC testing

This menus allows the selection of what value is to be put out of the DAC. The value to put out is entered. It can be any 16 bit hex value from 0 to ffff. The value may then be adjusted up or down by hitting the plus or minus keys respectively. In addition, new starting values may be specified and put out and then adjusted.

3.7) ADC testing

This menu allows the selection of bringing in values from the ADC. 12 bit values are brought in from the ADC and are displayed in hex format. A new ADC read is performed and it's corresponding value is displayed every time a return key is hit on the keyboard.

3.8) Running the frequency generation and analysis package

To create the frequency generation and analysis package:

(NOTE: must have appropriate definicon and custom hardware and software available to the PC in use.)

1) First, compile the source files needed (.c -> .i):

```
load -t c freq gen
```

```
load -t c plot1
load -t c g1
load -t c graph
```

2) Second, create the object files from the intermediate files (.i -> .obj):

```
load jcode freq gen.i
load jcode plot1.i
load jcode g1.i
load jcode graph.i
```

3) Third, link the files (.obj -> .e20):

```
load link20 freq_gen plot1 g1 graph  clib paslib
```

4) Now the package can be run (.e20 files are executable):

```
load freq gen
```

3.9) Results of development

The system which evolved proved to be highly flexible and utilitarian. After development was completed, the main usage of the package was to follow a chain of actions which tested handguns for their frequency responses to ultrasonic bombardment. Either the sweep mode, or previous knowledge as to what discrete frequencies were critical, was used to generate a wave which had the frequencies of interest present. If in sweep mode, the wave would be sent out with various delays and lengths in order to find out which delays and lengths evoked the greatest response from the target gun. Then, by analysing the frequency response of the target resonance, the frequencies of interest were more specifically addressed by switching to the discrete mode of frequency generation. A tailored single frequency, or compound, wave would then be generated and transmitted, and the received response would then be analyzed.

At this time the auto FFT mode in the data transfer portion of the system would be entered to allow the easy use of the developed tools in an iterative manner so that repeated tests could be performed with the same wave. This type of testing would be done when such factors as the position of the gun and the volume of the transmitted wave were under investigation. They would be varied while the characteristics of the wave itself would be held constant.

In an effective automatic ultrasonic gun detector this same sequence of events would be carried out, and would make automatic detection possible provided that reliable excitation patterns were obtained from the target.

3.10) Splicing in additional software functionality

The software package is flexible as to the addition of new menus, expanding the existing menus, and adding new functions. Note however, that a working piece of software is worth two (at least) software projects under development (or should we say, "in the bush"). Suffice it to say that it would behoove an ambitious user to keep a copy of the original, or other, working code around while performing extensions or remodeling the package.

The main way to add functions should be through the system of menu driven control. Basically, what happens is that:

A particular KEY is hit at the same time that a particular POSITION in the system is current and while certain PAST EVENTS have gone on dealing with that position.

The KEY hit is taken in and assigned a number, which right now for the purpose of menu driven control is:

down - 0
up - 1
right - 2
left - 3
end - 4
return - 5

This number is used as an offset into a portion of a structure of "action vectors". The offset gives a final pointer to a particular "action vector". What an action vector is, is a group of pointers to functions (the function names) with an associated count of those pointers to functions. A routine, called TAKE ACTION, takes as it's input the key number, gets the pointer to the action vector, and then executes the "count" number of functions in that action vector. In this way, the hit of a single key can cause a group of desired functions to be carried out.

Now, how does the system know which action vector to execute? The action vector being addressed is the action vector at the current system POSITION of course.

A POSITION is a structure containing the components of logical menu, row, and column. These components specify where in the system, logically, we are (i.e. the address). A structure of type POSITION INFO contains all of the action vectors. Using the position components to arrive at the action vectors, we also arrive at information specifying the physical addresses of where the menu fields are (i.e. where on the screen to put something). This physical information consists of absolute row, absolute column, and field length. So by having the information of POSITION, and a KEY, we have available the physical information of where on the screen to represent any changes desired, as well as a link to an action vector whereby we can carry out any set of functions desired. In short, we have a system for translating keystrokes and a graphic background into actions, or a menu driven operating system.

To modify or add functions-

Look at the structures of action vector and position info. Note that structures of the type position info contain a structure of the type action vector. The action vector contains a count and an array of functions.

```
struct action vector
{
int count;
void ( *func[max func dimen] ) ();
};


struct position info
{
int length;
int abs_row;
int abs_col;

struct action_vector   act vect [max key];
};
```

As the primary example, look at the actual structure used (you will most likely be modifying it). It's an array LOCATION, and is of the type POSITION INFO. A small excerpt is shown below. Notice the line across from /- down -/. It is the first action vector of this menu and position. 3 is the count. Unhilight, inc row, and hilight are the functions. When the key DOWN is hit and the position (menu,row,column) leads us to this particular position info structure, the 3 functions are carried out. If you wanted to run only the first two upon the reception of a DOWN key at this position, merely change the number to 2. If you want to run three routines you have written yourself (say one, two, and three) merely change the names to the names of the routines you wish to run. Upon the receipt of a down key when at this position, those three routines will be run in sequence.

```
                {
                f2_leng, sf1 row, box col+13, <-- LENGTH, ABS ROW, ABS COLUMN

/- down   -/   ,->    { { 3, { unhilight, inc row, hilight     } },
/- up     -/   |->      { 0 },
/- right  -/   |->      { 0 },
/- left   -/   |->      { 3, { unhilight, go to quit, hilight  } },
/- end    -/   |->      { 3, { unhilight, go to quit, hilight  } },
/- return -/   |->      { 2, { activate pos, menu select go    } } } } },
               |
               | ACTION VECTORS
```

Figure 3.2 - A group of action vectors

Note that the absolute physical information is shown just before the action vectors. By changing these values, routines such as hilight and unhilight will perform their duties at the newly specified screen locations and lengths.

More drastic remodeling-

To expand menus to more than 7,

add more than 5 functions to an action vector,

add more than 7 rows to a menu's choices,

add more than 2 columns to a menu's choices,

or to respond to more than 6 keys,

you must change the definitions for:

max_menu

max_row

max_col

max_key

max_func_dimen

and must also change the structure LOCATION to accommodate the new definitions. Be extremely careful in how the brackets "{" and "}" are used to define levels within the structure. The current layout of these brackets and the data within was found to be convenient. Remember, a "{" means going down a level in the structure while a "}" means coming up a level.

A complete listing of the code comprising the frequency generation and analysis package is included in appendix B.

IV RESEARCH

4.0) Transducers

The receiver was based on the Polaroid ultrasonic transducer. This is a capacitance based device which exhibits good sensitivity, but is very directional and does not have a flat frequency response across the range of 20 to 100 Khz. The frequency response can be compensated for electronically, but the directionality posed a sizeable problem. The directionality of most receiver traducers is related to the physical size of the actual receiver area. Generally, the larger the receiver surface area, the greater the sensitivity. But as the surface area increases, the more directional the device becomes. Because ideal receivers for this project would be omnidirectional, this poses a sizeable problem.

One method that could be used to overcome the directionality problem would be to use several receivers and position them in such a manner that when sound is radiated from a target (in a certain area), at least one of the receivers would be in the direct path of the signal. By using electronic switching devices, we could then sample all of the receivers, and determine which one would have the largest signal from the target. This poses a significant problem electronically in just determining which of the receivers has the most signal energy as compared to noise energy.

Another method the might minimize the directional effect would be to use the receivers in an array format and effectively AVERAGE them in frequency domain (the average must take place in the frequency domain not the time domain because the phase needs to be separated from the signal).

Noise in the frequency domain will average out to a known level leaving an averaged target value. The target value would then be the sum of many different receivers, thus providing more signal to noise ratio as compared to just one receiver. This method, however is likely to become very computationally intensive, where each receiver might need its own digital signal processor. More research into this area should be done.

Other areas of receiver interest would be to design a receiver that is extremely sensitive while remaining non-directional. Some effort was done in researching different receiver technologies, and none were found to be compatible with our requirements. Again we feel that much more work needs to be done on the receiver technology.

4.1) Target Chamber

Due to limited personnel, we were not able to work on designing a good target chamber. Consequentially we used the chamber material from the Phase II project. The chamber we used was not very good at dampening reverberation quickly especially at frequencies above 55 Khz. This forced us to use extra long trigger delays with the A/D converter, so that the original transmitted signal has died down below the noise floor. The longer the delays, the less of the target signal was present. With respect to the chamber design, this is probably the largest problem in the system to overcome. If however we could devise a chamber that would quickly dampen out the reverberation effects, this would relax other elements in the system [Quickly dampen means attenuating the reverberation to below the noise floor with in about 10 to 20 ms. We feel that this is very obtainable using the right materials and with proper construction of the chamber].

Our experiments indicated that the target would start to resonate almost immediately after the sound waves hit it. The amount of time it took to dampen out depended on several factors, such as gun placement (relative to the incident sound wave), gun material (Steel, High carbon steel, stainless steel, plastic, etc.), what was touching the gun (cloths, flesh, holster, etc.) and various other factors. All of these variables end up in reducing the amount of resonate time of the gun itself. For this reason, the sooner that we can sample the target signal the better signal to noise ratio. But because of the reverberation in the chamber, we were forced to wait until it died out (below the noise floor or threshold of the receiver) before we could actually sample the target signal. We feel that a chamber could be designed to limit the effect of reverberation such that the target signal could be sampled before most of its energy is expired. Again much more work in the area of chamber design needs to take place.

4.2) Time Delay Spectrometry

One method of removing the effects of reverberation is called Time Delay Spectrometry. This technique has been extensively developed by speaker manufacturers, in order to accurately measure the frequency response of a speaker as if it was in a free field. The free field implies that the only stimulus on the measurement device is coming directly from what is being measured. In other words, all reflections and reverberations are completely removed [A simple example of a way to take speaker measurements in a free field would be to suspend a speaker by a rope from a balloon a great distance above the ground and below the balloon. In this case there would be no sound reflections or reverberations (except that of the rope and the receiver itself).]. In gross terms, Time Delay Spectrometry can remove the effects of reflected and reverberation signals electronically.

This method evolves transmitting a sweep of frequencies at a known sweep rate, and then calculating the delay between the transmission of the sweep and the reception of the first reflected incidence of the sweep signal. Then by using heterodyning techniques, filter out all other frequencies except the incident frequency. Any reverberation frequencies at the receiver would be of a different frequency (because of the time delay and sweep rate) and therefore would be filtered out. This method was devised to input the reflected signal, not a secondary signal such as the resonating of a target. However, if the target is going to resonate at some particular frequency, then, it will absorb more energy at that particular frequency that all the others. In this case, the absorption of a particular frequency might be detectable rather than the ringing itself. We were not able to pursue this method in detail but feel that it can offer a possible solution to the reverberation problem that we have encountered.

Finally anther method that would not be prone to reverberation would be to excite the target using a frequency slightly below or above a resonant frequency. At the receiver, filter out the transmitting frequency by a very narrow notch filter. Because the notch filter is not at the point of resonance of the target, it should only filter out the transited signal and not the resonating signal of the target. In addition to the difficulty of designing a stable and accurately variable notch filter, we found through experimental data that very little energy is transferred to the target unless the resonant frequency is transmitted. Even when the transmitting frequency is .1% away from the resonant frequency, only about 20% of the maximum energy transfer takes place. These experiments confirmed the results of the previous test. Because of this, and the fact that the number of different frequencies needed to cover even a small number of the probable guns would be prohibitively large, we feel that this method would not be very practical.

4.3) Pattern Recognition

Preliminary studies were undertaken into pattern recognition techniques involving fuzzy sets and rough sets approaches. The reason for these particular approaches has to do with the nature of the data being processed and the speed with which the problem must be solved in order to have a detector which approaches anything like real time classification. The great variety of acoustic situations and the large numbers of variables which can affect those situations will cause any patterns in a frequency response from environments including a gun to exhibit a great deal of variability. Even the detection pattern for a particular type of gun would not be the same for different real guns of the same make and brand due to the differences in the guns themselves. Because of the variability in patterns, a way must be found to get the most information out of imprecise data. When working with imprecise data, the techniques of Zadeh, and many who have followed his general tenets of fuzzy logic systems (Zadeh)(Gupta), and the rough sets techniques of Pawlak, become likely candidates for use in classification schemes.

Very briefly, fuzzy set theory extends the mathematics of predicate calculus and probability theory to include imprecise measurements, data, and classifications. An object may be classified with a degree of membership to a set. In one case a pattern could be said to belong to the set of patterns elicited from Smith and Wesson .357's with a degree of membership of 0.78. A law in an expert system could describe an example of this sort as a pattern which was "likely" that of a Smith and Wesson .357 where the modifier "likely" carried with it the numerical coefficient 0.78. The mathematics for objects and sets described in terms of fuzzy logic, though still relatively new, is well worked out and used in several working systems.

Another system, that of rough sets, looks very promising because of the speed with which it can work in a classification scheme. In rough sets theory an information system is defined (Pawlak),

$$S = (U,P,V,F)$$

where S is the system,

U is the universe of objects,

P is the set of attributes,

V is the union of the domains of all p's in P,

F is U x P -> V is a total function such that

F(x,q) is in Vq for every q in Q and every x in U. It is called the information function

Suppose we have a finite set of objects U which make up the universe (possibly handgun excitation patterns). Elements of U are training examples. U is the training set. Suppose that an expert breaks U into classes {x1,...xn} based on absolute knowledge of the objects. Suppose a learning agent tries to characterize the object of U in terms of attributes from P. The descriptions of objects, based on P, represents the knowledge of the learning agent concerning objects in U.

To what degree can the learning agent's knowledge classify the objects according to the available attributes to fit the expert's classification? What is desired is a classification algorithm which provides the expert's classification based on the attributes of objects.

f(x) is the learning agent's knowledge about x in S (since it crosses U x P -> V). Let's extend the information system by adding e, the attribute describing the expert's classification. (i.e. e* = {x1,x2,....xn}, e* is the classification of the objects by the expert, so we get classes 1..n)

The new information system S' = (U,Q,V',F'); Q = P union {e}, P intersect {e} = 0, is created where the expert's knowledge on x in S is the class to which x belongs. The problem on static learning becomes the question of whether the classification e* is P-definable. If e* is P-definable then the algorithm to "learn" classification e* exits.

e* is P-definable iff P -> e; e depends on the set of attributes P.

Suppose r* is supplied by an expert. Can classification be expressed in terms of p and q?

p	q	r
1	0	2
0	1	1
2	0	0
1	1	0

because the dependance P = {p,q} -> r holds the learning algorithm exists.

it is (p:= 1) (q:= 0) => (r:= 2)

(p:= 0) => (r:= 1)

(p:= 2) + (p:= 1)(q:= 1) => (r:= 0)

Often though the expert's knowledge (the extra attribute) will not be definable by the attributes. In this case approximation is possible. The classification of some objects correctly is possible. Rough sets theory provides for coefficients which show to what degree such approximations are valid in what is termed Accuracy and Quality, where quality is a function of what part of the objects may be classified correctly and accuracy is what part of decisions can be correct.

Consider the information system:

U	p	q	r
x1	1	0	2
x2	0	1	1
x3	2	0	0
x4	1	0	2
x5	1	0	0
x6	0	1	1
x7	2	0	0
x8	1	0	0
x9	0	1	1
x10	2	0	0
x11	1	0	0
x12	1	0	2

assume r is the expert's knowledge, so that r* is the experts classification.

The system with respect to all attributes is

U/Q~	p	q	r
z1	1	0	2
z2	0	1	1
z3	2	0	0
z4	1	0	0

The atoms of the system are,

z1 = {x1,x4,x12}

z2 = {x2,x6,x9}

z3 = {x3,x7,x10}

z4 = {x5,x8,x11}

where $U/Q^{\sim}$ means U with respect to the equivalence relation of Q denoted $Q^{\sim}$.

The classification r* is not (p,q) definable so we will approximate r* by (p,q).

Classes of classification r* (equivalence classes of relation $r^{\sim}$)

$A1 = \{x1, x4, x12\}$

$A2 = \{x2, x6, x9\}$

$A3 = \{x3, x5, x7, x8, x10, x11\}$

Equivalence classes of relation $(p,q)^{\sim}$

$B1 = \{x1, x4, x5, x8, x11, x12\}$

$B2 = \{x2, x6, x9\}$

$B3 = \{x3, x7, x10\}$

The concept of upper and lower approximation is now applied to the classifications of the relations.

Let P = {p,q}. Then the following sets are the lower P approximations of r*.

$\_PA1 = 0$

$\_PA2 = B2$

$\_PA3 = B3$

and the following sets are the upper P approximations of r*.

-PA1 = B1

-PA2 = B2

-PA3 = B1 U B3

We have arrived at the situation where A1 is internally P nondefinable, A2 is P definable, and A3 is roughly P definable. Or, stated another way, A2 can be learned fully, A3 can be learned roughly, while for A1 it is impossible to classify correctly x1,x4,x12 by observing features expressed by p and q (though negative instances may be learned anyway).

We have the non-deterministic classification algorithm

(p:= 1)(q:= 0) => (r:= 2) + (r:= 0)

(p:= 2)(q:= 0) => (r:= 0)

(p:= 0)(q:= 1) => (r:= 1)

Because of the simplicity with which classification algorithms are specified the algorithms can run very quickly. If the algorithm does not provide enough accuracy or quality, more attributes may be added to the information system until acceptable levels are reached.

Some researchers (Gupta)(Prade) have combined the fuzzy set theory with rough set classification schemes to make use of the benefits of each. The ability to deal with varied and imprecise data makes these methods very promising candidates for use in the automatic ultrasonic handgun detector.

4.4) FFT systems

The reason for developing an FFT system is that we have to take data in the time domain, which is what we receive through our transducers and A to D converters, and change it to the frequency domain so that we can analyze it in a meaningful form in the search for patterns representative of handguns. The Fast Fourier Transform gives the same results as the Discrete Fourier Transform since it is mathematically the same transform. However, for computations done on machinery, often the case is true that how the calculations are done has an enormous impact on the speed and accuracy of the obtained results. It was necessary to obtain our results very quickly since the object of our research was a real time analysis. Therefore, we decided to investigate the field and to find the proper FFT algorithm for our application, first in software, and eventually committing the algorithm to hardware where it would be fast enough for our needs.

4.5) Software FFT algorithms

A typical straightforward program for the direct calculation of the DFT might look something like (written in C):

```
q = 6.2831 / n;

for (j= 0; j< n; j++)
 {
        a[j] = 0;
        b[j] = 0;

        for (i= 0; i< n; i++)
         {
                a[j] += x[i] * cos(q*i*j) + y[i] * sin(q*i*j);
                b[j] += y[i] * cos(q*i*j) - x[i] * sin(q*i*j);
         }
 }
```

Figure 4.0 - A Straightforward Discrete Fourier Transform

where n is the number of points for which the transform is being done, q is the degree to which the unit circle is broken down for the trigonometric factors to be applied at proper intervals, arrays a[] and b[] are the real and imaginary parts of the frequency domain results of the computation, and arrays x[] and y[] are the real and imaginary parts of the time domain data which go into the transform.

There is nothing inherently wrong about the piece of code written above. It's straightforward, simple, and short. It's just slow, performing as an order n squared algorithm (note the nested loops which run from 0 to n-1). Improvements can be made by adding a look up table for the trigonometric calculation, by doing the trigonometric multiplications outside of the inner loop and saving the results for the multiple usage within the inner loop. Other modifications can be made which make use of the fact that the trigonometric functions repeat. However, even though useful, none get rid of the overwhelming disadvantage of the n squared order of the algorithm itself.

The big break in this problem comes from the introduction of the Cooley-Tukey FFT algorithm which achieved an n log n order of computational complexity. Essentially all FFT algorithms make use of the same techniques as the Cooley-Tukey algorithm. The basis behind the reductions in calculation for their algorithm lies in the building up of parts of larger calculations with smaller calculations while using the smaller parts along the way, and generally in coming up with a systematic way to use portions of calculations where and when they are needed instead of recalculating them whenever they are needed. The partial results must of course be stored in places where they can be easily retrieved at the proper moment during the execution of the algorithm in a real system.

A typical Cooley-Tukey program for the radix-2 calculation of the DFT might look something like (written in C):

```
n2 = n;
for (k=1; k <= m; k++)
 {
        n1 = n2;
        n2 = n2/2;
        e = 6.283185307179586 / (n1);
        a = 0;

        for (j=1; j <= n2; j++)
         {
                c = cos(a);
                s = sin(a);
                a = j * e;

                for (i=j; i <= n; i += n1)
                 {
                        /* the butterfly */

                        l = i + n2;
                        xt  = x[i-1] - x[l-1];
                        x[i-1] = x[i-1] + x[l-1];
                        yt= y[i-1] - y[l-1];
                        y[i-1]= y[i-1] + y[l-1];

                        /* the twiddle factor multiplications */

                        x[l-1] = c*xt + s*yt;
                        y[l-1]= c*yt - s*xt;
                 }
         }
 }
```

Figure 4.1 - A radix 2 Cooley-Tukey FFT program in C

Because of the way the flow diagrams look which express this building up of calculations along the way to the final results of the algorithm, and especially the way in which the smallest portion of the transform is represented, the method has been called the butterfly transform.

Figure 4.2 - A single Radix 2 Butterfly

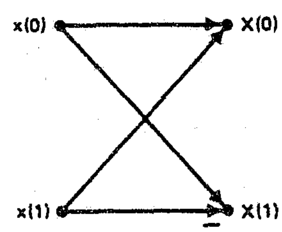
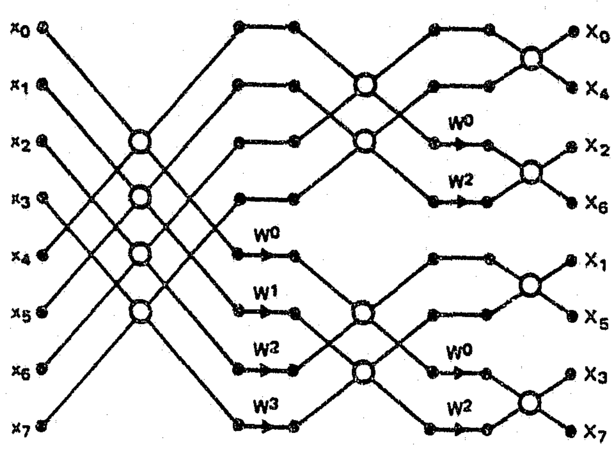


Figure 4.3 - A flowgraph of a length 8 Radix 2 FFT



There are many differing ways in which to implement the basic idea behind the FFT. There are decimation in time algorithms and decimation in frequency algorithms which specify in which direction the complexity of the stages of the butterflies (algorithm subcomponents) increases. There are different radices for the size of the smallest subcomponents, or butterflies. There are schemes for efficiently applying the trigonometric factors, commonly called "twiddle factors", skipping the multiplications by unity and shortening the process when the factor is zero. The most relevant of these differences comes with the use of radix as choice of radix has the most to give in terms of reducing computational operatons.

Number of real multipies and adds for different complex one butterfly FFT algorithms

	m	n	multiplies	adds	mults+adds
radix = 2					
	1	2	4	6	10
	2	4	16	24	40
	3	8	48	72	120
	4	16	128	192	320
	5	32	320	480	800
	6	64	768	1152	1920
	7	128	1762	2688	4480
	8	256	4096	6144	10240
	9	512	9216	13824	23040
	10	1024	20480	30720	51200
	11	2048	45056	67584	112640
	12	4096	98304	147456	245620
radix = 4					
	1	4	12	22	34
	2	16	96	176	272
	3	64	576	1056	1632
	4	256	3072	5632	8704
	5	1024	15360	28160	43520
	6	4096	73728	135168	208896
radix = 8					
	1	8	32	66	98
	2	64	512	1056	1568
	3	512	6144	12672	18816
	4	4096	65536	135168	200704
radix = 16					
	1	16	80	178	258
	2	256	2560	5696	8256
	3	4096	61440	136704	198144
*					

Note: m is the power to which the radix is raised.
n is the number of points in the FFT.

Figure 4.4 (Burris)

The radix refers to the size of the stages making up the individual butterflies. They are frequently powers of two, although others are possible. The number of points in a transform is related to the radix by the formula:

n = r**m

where n is the number of points in the transform, r is the radix, and m is the number of dimensions, or stages required.

We found that using an algorithm with a radix of four gave us a savings in speed that was worth the problems encountered in the additional complexity of code and the inconvenience inherent in the limitations on the length of our FFTs which were then limited to being the size of a power of four. Other improvements, such as the use of trig look up tables, did very little in comparison to the radix changes in decreasing the time for algorithm completion.

V CONCLUSIONS AND RECOMMENDATIONS

In phase III/1 of this research program:

- a hardware system for the field test unit was designed and implemented using off the shelf components for digital electronics and for all of the amplifiers and transducers,

- a software package was developed which allows easy access to all of the functions supported by the hardware, in a flexible and extendable multilayered menu driven package,

- the total hardware/software system formed a package with the ability to be used as a data acquisition system by people with little or no training,

- the hardware/software system achieved the ability for automatic iterative looping of the functions for configuring a specified wave, placing the digital points of the wave with any looping information present in the D to A RAM, transmitting the specified wave, receiving incoming sounds at a designated time delay after the transmission has begun, placing a digitized representation of the incoming sound in the A to D RAM, and finally analyzing the data received in it's raw form, or after transformation by fast fourier techniques into the frequency domain, and displaying the desired data graphically in a form which may be chosen by the user and is easy to understand,

- a radix four Fast Fourier Transform algorithm was chosen and implemented in software to transform the time domain data into the frequency domain. Preliminary investigations into the incorporation of this technique into hardware have begun in order to increase the speed of this task. The increase of speed is important here because the number of points taken in an FFT is linearly proportional to the resolution in the frequency domain while it is n log n proportional to the number of computations necessary. A large number of points is needed to gain great resolution which tends to take a lot of computation. Since speed is of the utmost importance in a real time system of detection, this topic is worthy of future development.

- development of a Time Delay Spectrometry system was begun in order to eliminate the need for a long wait after the transmission of sound before receiving. The reason for waiting has been to let the transmitted sound, and it's reverberations, die out before listening for the sounds generated by the modal excitation of the target handguns. The TDS system has the task of making this wait unneccessary by using frequency sweep and time information to pick out the valid information while the noise in the environment is still at high levels. The sound containing the desired information is at higher levels when obtained before a delay allows the sound to diminish due to natural decay,

- investigations into pattern recognition techniques involving imprecise forms of data and data partitioning have begun. The speed with which the results must be obtained, and the nature of the data received which will naturally exhibit a great deal of variability, makes these types of pattern recognition techniques the logical choice for the project at hand. Also such techniques allow for easy implementation of threshold setting which is important in a project involving security.

- research into phased array polaroid transducers was begun. Phased arrays are to be used to allow the use of individual transducers which are high in sensitivity but are poor in impartiality to directionality. By using a large array of these transducers, the directionality problem can be overcome by averaging the results of many transducers, some of which will be in a correct position for good reception.

- materials testing was begun to investigate various types of material as to the ability of these materials to stop sound from penetrating and thus hampering the detection of concealed handguns. Preliminary testing shows that porous substances such as stryrofoam and clothing are not good sound insulators whereas more solid substances such as wood are better at stopping high frequency sounds from penetrating.

It is recommended that the phase III effort be continued to completion of a useful Field Test unit. To accomplish this:

- the development of transducers, or of a transducer system, which is sensitive to low levels of sound as well as to sounds coming from directionally different axis should be carried out. The critical importance of the transducers in the success of this project cannot be overemphasized,

- the development of pattern recognition techniques which will work with rules allowing for imprecise data partitioning should be carried out and should allow the specification of levels for threshold triggering,

- the technique of Time Delay Spectroscopy should be further developed and incorporated into the design of the system to allow a reduction in the dampening of the excited sound due to the time delay currently neccessary,

- the FFT algorithm should be implemented in hardware so as to increase the speed to the point where the calculations involved take up a negligible time in proportion to a total iteration of the detection loop, and finally,

- that the system make use of the above mentioned developments in a package which is usable by people with no training or conception of the technical aspects of the underlying operation.

BIBLIOGRAPHY

Burris, C. S., Parks, T. W., **DFT/FFT and Convolution Algorithms**, John Wiley & Sons, New York, 1985.

Crawford, Harold B., Williams, Joseph, **Handbook of Electronics Calculations**, Mcgraw Hill, 1979.

Gupta, Madran M., et. al., **Fuzzy Automata and Decision Processes**, North Holland Publishing Co., Amsterdam, The Netherlands, 1977.

Jackson, Leland B., **Digital Filters and Signal Processing**, Kluwer Academic Publishers, 1986.

Kernighan, B. W., Ritchie, D. M., **The C Programming Language**, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

Manassewitsch, Vadim, **Frequency Synthesizers, Theory and Design**, Wiley-Interscience, 1987.

Pawlak, Z., "Rough Classification", Int. J. Man-Machine Studies, 20, 1984, pp. 469-483.

Pawlak, Z., "On Learning - A Rough Sets Approach", unpublished, 1986.

Rabiner, Lawrence R., Gold, Bernard, **Theory and Application of Digital Signal Processing**, Prentice-Hall, Englewood Cliffs, NJ, 1975.

Van Valkenburg, M. E., **Analog FIlter Design**, CBS College Publishing, 1982.

Williams, Charles S., **Designing Digital Filters**, Prentice-Hall, Englewood Cliffs, NJ, 1986.

Zadeh, Lofti Asker, et. al., **Fuzzy Sets and their Applications to Cognitive and Decision Processes**, Academic Press, New York, 1975.

-*-*-*- APPENDIX A -*-*-*-

GAL listings

```
GAL16V8   7/21/87
FILENAME: 'ut3_dsak.pld'          DSACK* timing generator.
AUTHOR:   James Vig Sherrill

;
;       This will generate the 2 DSACK signals back to the processor. It
;     also contains a watch dog timer to prevent any hang up in the system
;     if an incorrect memory position was decoded. The watchdog will take
;     place only after the HIT line has been activated, and 32 CPU clock
;     cycles have gone by.
;


CLK                     ;1
GP_DA!                  ;2
NCTO                    ;3
PT_DA!                  ;4
NCT1                    ;5
HIT!                    ;6
RW                      ;7
AS!                     ;8
NC2                     ;9
GND                     ;10  GND
OE                      ;11
CNT0                    ;12
CNT1                    ;13
DSACK1!                 ;14
RD_WR                   ;15
DET                     ;16
CNT2                    ;17
CNT3                    ;18
AS_RST!                 ;19
PWR                     ;20 is VCC

;
;Must have a 2 line break between pin descriptions and equations.
;

CNT0       := /HIT! * /CNT0

CNT1       := /HIT! *  CNT0 * /CNT1
            + /HIT! * /CNT0 *  CNT1

CNT2       := /HIT! * /CNT2 *  CNT1 *  CNT0
            + /HIT! *  CNT2 * /CNT1
            + /HIT! *  CNT2 * /CNT0

CNT3       := /HIT! * /CNT3 *  CNT2 *  CNT1 *  CNT0
            + /HIT! *  CNT3 * /CNT2
            + /HIT! *  CNT3 * /CNT1
            + /HIT! *  CNT3 * /CNT0

/DSACK1!   := /HIT! *  CNT3          * /GP_DA! * PT_DA!    ;Wait 8 cycles fast
            + /HIT! *  CNT3 *  CNT2 * /PT_DA!              ;Wait 12 cycles slow
            + /HIT! *  CNT3 *  CNT2 *  CNT1                ;Watch dog timer

/DE!       := /HIT! *  CNT1                         ;Delay because CNT0 is not
            + /HIT! *  CNT2                         ;  included.
```

```
         + /HIT! *  CNT3

/RD_WR     := /HIT! * /RW *  CNT0  * /CNT3 * /GP_DA!
            + /HIT! * /RW *  CNT1  * /CNT3 * /GP_DA!
            + /HIT! * /RW *  CNT2  * /CNT3 * /GP_DA!     ;*********
            + /HIT! * /RW * /CNT3          * /PT_DA!
            + /HIT! * /RW *  CNT3  * /CNT2 * /PT_DA!


/AS_RST!   := /DSACK1! * AS!
```

DESCRIPTION

The CNT0 - CNT3 are the counter lines. When HIT! is low, these lines will count CPU cycles. When HIT! is high, they are reset to 0.

```
*  If the board has been HIT, and neither a PT_DA! or a GP_DA! respond, then
*  the watch dog timer will generate DSACK1! and AS_RST! after 16 CPU cycles.
*  If a  PT_DA or a GP_DA has happened, then DSACK! will be generated after 4
*  CPU cycles.
*
*  The PT_DA and GP_DA have been removed.
```

DE! is the data enable for the Bidirectional latches that are on the CPU DATA lines. DE! Becomes active 1 CPU cycle after HIT! becomes active. It stays active as long as HIT! is active.

RD/WR! is the Read/Write line. This signal is based on the RW line generated from the CPU. If RW is high (CPU reading from external RAM) then RD/WR! line will be high. If RW is low (CPU Writing to external RAM) then RD/WR! will go low when HIT! goes low and stay low for 3 CPU cycles. It will go high 1 CPU cycle BEFORE DSACK is generated.

AS_RST! is the AS reset line. It is used to reset the FF after AS set it. It is gated with the CPU AS! and DSACK. This is done to make sure that DSACK is asserted long enough (Once DSACK is asserted, it stays asserted until AS! goes inactive.

```
GAL20V8    7/21/87
FILENAME: 'ut3_cnt4.pld'       4 bit extension counter with address decodes
AUTHOR:   James Vig Sherrill

;
;       The counter is a syncrounous 4 bit counter with syncrounuous load.
;    It is designed to be used with a ut3_cnt7.pld and a ut3_cnt6.pld to
;    yield a total of 17 address lines. This gal provides bits
;    Address[16:13]. Also it decodes the outputs Address[16:15] into 4
;    seperate address decodes.
;
;
;    /CS0! =    /A16 * /A15
;    /CS1! =    /A16 *  A15
;    /CS2! =     A16 * /A15
;    /CS3! =     A16 *  A15
;
;

COUNT                        ;PIN 1 is COUNTER CLOCK pin table begins on line 5
NC0                          ;Not used
LOAD                         ;PIN 3 is LOAD
AD_13                        ;Address input 13
AD_14                        ;Address input 14
AD_15                        ;Address input 15
AD_16                        ;Address input 16
NC1                          ;
NC2                          ;
NC3                          ;
C_IN                         ;Carry IN
GND                          ;GND
OUT_EN                       ;Output Enable (ACTIVE LOW)
NC4                          ;NOT USED
C_13                         ;Address 13
C_14                         ;Address 14
C_15                         ;Address 15
C_16                         ;Address 16
CS0!                         ;Chip Select 0 ACTIVE LOW
CS1!                         ;Chip Select 1 ACTIVE LOW
CS2!                         ;Chip Select 2 ACTIVE LOW
CS3!                         ;Chip Select 3 ACTIVE LOW
NC5                          ;NOT USED
PWR                          ;Pin 24 power

;
;Must have a 2 line break between pin descriptions and equations.
;

C_13      :=  LOAD * AD_13
            + /LOAD * /C_13 *  C_IN
            + /LOAD * /C_IN *  C_13

C_14      :=  LOAD * AD_14
            + /LOAD * /C_14 *  C_13 * C_IN
            + /LOAD *  C_14 * /C_13 * C_IN
            + /LOAD * /C_IN *  C_14
```

```
C_15       :=  LOAD * AD_15
             + /LOAD * /C_15 *  C_14 *  C_13 *  C_IN
             + /LOAD *  C_15 * /C_14        *  C_IN
             + /LOAD *  C_15 * /C_13        *  C_IN
             + /LOAD * /C_IN *  C_15

C_16       :=  LOAD * AD_16
             + /LOAD * /C_16 *  C_15 *  C_14 *  C_13 *  C_IN
             + /LOAD *  C_16 * /C_15 *                  C_IN
             + /LOAD *  C_16 * /C_14 *                  C_IN
             + /LOAD *  C_16 * /C_13 *                  C_IN
             + /LOAD * /C_IN *  C_16

/CS0!      =  /C_16 * /C_15
/CS0!.OE   =  /OUT_EN

/CS1!      =  /C_16 *  C_15
/CS1!.OE   =  /OUT_EN

/CS2!      =   C_16 * /C_15
/CS2!.OE   =  /OUT_EN

/CS3!      =   C_16 *  C_15
/CS3!.OE   =  /OUT_EN
```

DESCRIPTION

```
GAL20V8    7/21/87
FILENAME: 'ut3_cnt4.pld'        4 bit extension counter with address decodes
AUTHOR:    James Vig Sherrill

;
;       The counter is a synchronous 4 bit counter with synchronous load.
;     It is designed to be used with a ut3_cnt7.pld and a ut3_cnt6.pld to
;     yield a total of 17 address lines. This gal provides bits
;     Address[16:13]. Also it decodes the outputs Address[16:15] into 4
;     separate address decodes.
;
;
;     /CS0! =     /A16 * /A15
;     /CS1! =     /A16 *  A15
;     /CS2! =      A16 * /A15
;     /CS3! =      A16 *  A15
;
;

COUNT                       ;PIN 1 is COUNTER CLOCK pin table begins on line 5
NC0                         ;Not used
LOAD                        ;PIN 3 is LOAD
AD_13                       ;Address input 13
AD_14                       ;Address input 14
AD_15                       ;Address input 15
AD_16                       ;Address input 16
OUT_EN1                     ;
NC2                         ;
NC3                         ;
C_IN                        ;Carry IN
GND                         ;GND
OUT_EN2                     ;Output Enable (ACTIVE LOW)
NC4                         ;NOT USED
C_13                        ;Address 13
C_14                        ;Address 14
C_15                        ;Address 15
C_16                        ;Address 16
CS0!                        ;Chip Select 0 ACTIVE LOW
CS1!                        ;Chip Select 1 ACTIVE LOW
CS2!                        ;Chip Select 2 ACTIVE LOW
CS3!                        ;Chip Select 3 ACTIVE LOW
NC5                         ;NOT USED
PWR                         ;Pin 24 power

;
;Must have a 2 line break between pin descriptions and equations.
;


C_13      :=  LOAD * AD_13
          +  /LOAD * /C_13 *  C_IN
          +  /LOAD * /C_IN *  C_13

C_14      :=  LOAD * AD_14
          +  /LOAD * /C_14 *  C_13 * C_IN
          +  /LOAD *  C_14 * /C_13 * C_IN
          +  /LOAD * /C_IN *  C_14
```

```
C_15      :=  LOAD * AD_15
            + /LOAD * /C_15 *  C_14 *  C_13 *  C_IN
            + /LOAD *  C_15 * /C_14           *  C_IN
            + /LOAD *  C_15 * /C_13           *  C_IN
            + /LOAD * /C_IN *  C_15

C_16      :=  LOAD * AD_16
            + /LOAD * /C_16 *  C_15 *  C_14 *  C_13 *  C_IN
            + /LOAD *  C_16 * /C_15 *                  C_IN
            + /LOAD *  C_16 * /C_14 *                  C_IN
            + /LOAD *  C_16 * /C_13 *                  C_IN
            + /LOAD * /C_IN *  C_16

/CS0!      =  /C_16 * /C_15
/CS0!.OE   =  /OUT_EN1

/CS1!      =  /C_16 *  C_15
/CS1!.OE   =  /OUT_EN1

/CS2!      =   C_16 * /C_15
/CS2!.OE   =  /OUT_EN1

/CS3!      =   C_16 *  C_15
/CS3!.OE   =  /OUT_EN1
```

```
GAL20V8    7/29/87
FILENAME: 'ut3_da1.pld'       The timing generator for all the D/A circuits
AUTHOR:    James Vig Sherrill

;
;
;          This GAL will generate all the timing necessary to run the D/A
;        section. It is a clocked device. The signal inputs and outputs are as
;        follows.
;
;
;        CNT0-CNT3        The count lines that all the timing is based upon.
;        These lines count from 0000 to 1111 (16 total states).
;
;        D0,D1            Inputs from the Definicon controller.
;
;
;

CLK                      ;PIN 1     (IN)
HIT                      ;2         (IN)
LOAD                     ;3         (IN)
RUN                      ;4         (IN)
CONV_IN                  ;5         (IN)
CNT0                     ;6         (IN)
CNT1                     ;7         (IN)
CNT2                     ;8         (IN)
CNT3                     ;9         (IN)
D0                       ;10        (IN)
D1                       ;11        (IN)
GND                      ;12        GND
OE                       ;13        (IN)
NC                       ;14        (IN)
STOP!                    ;15        (OUT)
C_END                    ;16        (OUT)
NC                       ;17        (OUT)
CNT_CLK                  ;18        (OUT)
AD_OE!                   ;19        (OUT)
AD_LD                    ;20        (OUT)
RAM_OE!                  ;21        (OUT)
RAM_WR!                  ;22        (OUT)
RESET!                   ;23        (IN)
PWR                      ;24        POWER

;
;

/STOP!          := /CNT3 *  CNT2 *  CNT1         *  D0 *  RUN *  HIT
                 + /CNT3 *  CNT2 *  CNT1         *  LOAD
                 + /CNT3 *  CNT2 *  CNT1         *  /RESET!

C_END           :=  CNT3 *  CNT2 *  CNT1 *  CNT0

CNT_CLK         := /CNT3 * /CNT2 *  CNT1         *  LOAD
                 +  CNT3 *  CNT2 * /CNT1 *  CNT0 *  RUN

/AD_OE!         :=  CNT3                         *  RUN *  HIT
```

```
                 +   CONV_IN

AD_LD            := /CNT3                    *  RUN *  HIT
                 +   CONV_IN

/RAM_OE!         := /CNT3 * RUN * HIT
 RAM_OE!.OE      =  /CNT3 * RUN * HIT

/RAM_WR!         :=  CNT3  * /CNT2  * RUN * HIT
 RAM_WR!.OE      =   CNT3  * /CNT2  * RUN * HIT
```

```
GAL20V8    7/21/87
FILENAME: 'ut3_cnt7.pld'        7 bit general purpose counter
AUTHOR:    James Vig Sherrill

;
;       The counter is a synchronous 7 bit counter with synchronous load.
;    The carry_out bit becomes valid only when bits 0 through 6 are 1. There
;    is no CARRY IN bit used, this will ALWAYS count or load.
;

COUNT                        ;Clock
NCLR                         ;NOT USED (Possibly used as a clear)
LOAD                         ;LOAD strobe
LD_0                         ;load input 0 LSB
LD_1                         ;load input 1
LD_2                         ;load input 2
LD_3                         ;Load input 3
LD_4                         ;Load input 4
LD_5                         ;Load input 5
LD_6                         ;Load input 6 MSB
NOT                          ;NOT USED
GND                          ;GND
OUT_EN                       ;Output Enable (ACTIVE LOW)
NCO                          ;NOT USED
C_0                          ;Count 0
C_1                          ;Count 1
C_2                          ;Count 2
C_3                          ;Count 3
C_4                          ;Count 4
C_5                          ;Count 5
C_6                          ;Count 6
C_OUT                        ;Carry OUT
NCO                          ;Not used input
PWR                          ;PIN 24 IS VCC

;
;Must have a 2 line break between pin descriptions and equations.
;

C_0     :=  LOAD * LD_0             ;LOAD
        +  /LOAD * /C_0             ;TOGGLE

C_1     :=  LOAD * LD_1                     ;LOAD
        +  /LOAD * /C_1 *  C_0              ;TOGGLE
        +  /LOAD *  C_1 * /C_0              ;HOLD


C_2     :=  LOAD * LD_2                             ;LOAD
        +  /LOAD * /C_2 *  C_1 *  C_0               ;TOGGLE
        +  /LOAD *  C_2 * /C_1                      ;HOLD
        +  /LOAD *  C_2 * /C_0                      ;HOLD

C_3     :=  LOAD * LD_3                                     ;LOAD
        +  /LOAD * /C_3 *  C_2 *  C_1 *  C_0                ;TOGGLE
        +  /LOAD *  C_3 * /C_2                              ;HOLD
        +  /LOAD *  C_3 * /C_1                              ;HOLD
        +  /LOAD *  C_3 * /C_0                              ;HOLD
```

```
C_4     :=  LOAD * LD_4                                          ;LOAD
        +  /LOAD * /C_4 *  C_3 *  C_2 *  C_1 *  C_0              ;TOGGLE
        +  /LOAD *  C_4 * /C_3                                   ;HOLD
        +  /LOAD *  C_4 * /C_2                                   ;HOLD
        +  /LOAD *  C_4 * /C_1                                   ;HOLD
        +  /LOAD *  C_4 * /C_0                                   ;HOLD

C_5     :=  LOAD * LD_5                                                   ;LOAD
        +  /LOAD * /C_5 *  C_4 *  C_3 *  C_2 *  C_1 *  C_0                ;TOGG
EL
        +  /LOAD *  C_5 * /C_4                                            ;HOLD
        +  /LOAD *  C_5 * /C_3                                            ;HOLD
        +  /LOAD *  C_5 * /C_2                                            ;HOLD
        +  /LOAD *  C_5 * /C_1                                            ;HOLD
        +  /LOAD *  C_5 * /C_0                                            ;HOLD

C_6     :=  LOAD * LD_6                                                   ;LOAD
        +  /LOAD * /C_6 *  C_5 *  C_4 *  C_3 *  C_2 *  C_1 *  C_0         ;TOGG
EL
        +  /LOAD *  C_6 * /C_5                                            ;HOLD
        +  /LOAD *  C_6 * /C_4                                            ;HOLD
        +  /LOAD *  C_6 * /C_3                                            ;HOLD
        +  /LOAD *  C_6 * /C_2                                            ;HOLD
        +  /LOAD *  C_6 * /C_1                                            ;HOLD
        +  /LOAD *  C_6 * /C_0                                            ;HOLD

C_OUT   = /LOAD *  C_6 *  C_5 *  C_4 *  C_3 *  C_2 *  C_1 *  C_0
C_OUT.OE = PWR
```

```
GAL20V8    7/29/87
FILENAME: 'ut3_da1.pld'        The timing generator for all the D/A circuits
AUTHOR:   James Vig Sherrill

;
;
;           This GAL will generate all the timing necessary to run the D/A
;       section. It is a clocked device. The signal inputs and outputs are as
;       follows.
;
;
;       CNT0-CNT3       The count lines that all the timing is based upon.
;       These lines count from 0000 to 1111 (16 total states).
;
;
;


CLK                         ;PIN 1    (IN)
C_OUT                       ;2        (IN)    Carry Out from LOOP COUNTER
LL                          ;3        (IN)    Indicating a Loop Load is running
AL                          ;4        (IN)    Indicating an Address Load is running
CNVT_IN                     ;5       (IN)    Indicating a CONVERT is running
CNT0                        ;6        (IN)    LSB of counter
CNT1                        ;7        (IN)
CNT2                        ;8        (IN)
CNT3                        ;9        (IN)    MSB of counter
D0                          ;10       (IN)    Data line D0, indicates a STOP bit
D1                          ;11       (IN)    Data line D1, indicates a LOOP bit
GND                         ;12       GND
OE                          ;13       (IN)
RUN                         ;14       (IN)    Running the D/A Converter
LOOP_CK                     ;15       (OUT)   Clocks the LOOP Counter
ADD_LD                      ;16       (OUT)   Loads the Address Counter
ADD_CLK                     ;17       (OUT)   Clocks the Address Counter
ADD_OE!                     ;18       (OUT)   Enables the Address Counters outputs
NC!                         ;19       (OUT)   Not used
CNVT_OUT                    ;20       (OUT)   Convert signal to actual D/A latches
STOP!                       ;21       (OUT)   Resets RUN on D0 or RESET or AL or LL
C_END                       ;22       (OUT)   A pulse always on CNT = 15
NC0                         ;23       (IN)    NOT USED
PWR                         ;24       POWER

;
;

C_END          :=  CNT3 *  CNT2 *  CNT1 *  CNT0    ; CNT = 15

CNVT_OUT       := /CNT3 * /CNT2 * RUN
                + CNVT_IN

ADD_CLK        :=  CNT3 * /CNT2 *  CNT1 * /CNT0 *  RUN            ;RUN * 10
                +  CNT3 * /CNT2 *  CNT1 * /CNT0 *  AL             ;AL  * 10

ADD_LD         :=  CNT3 * /CNT2                 *  RUN *  D1 * /C_OUT
                +  CNT3 * /CNT2                 *  AL

LOOP_CK        := /CNT3 *  CNT2 *  CNT1 * /CNT0 *  RUN *  D1 * /C_OUT
```

```
                 + /CNT3 *  CNT2 *  CNT1 *  CNTO *  RUN *  D1 * /C [illegible]
                 + /CNT3 *  CNT2 *  CNT1 * /CNTO *  LL
                 + /CNT3 *  CNT2 *  CNT1 *  CNTO *  LL

/STOP!          := /CNT3 *  CNT2 *  CNT1 * /CNTO *  RUN *  DO
                 + /CNT3 *  CNT2 *  CNT1 *  CNTO *  RUN *  DO
                 + /CNT3 *  CNT2 *  CNT1 * /CNTO *  LL
                 + /CNT3 *  CNT2 *  CNT1 *  CNTO *  LL
                 + /CNT3 *  CNT2 *  CNT1 * /CNTO *  AL
                 + /CNT3 *  CNT2 *  CNT1 *  CNTO *  AL
                 + /CNT3 *  CNT2 *  CNT1 * /CNTO *  CNVT IN
                 + /CNT3 *  CNT2 *  CNT1 *  CNTO *  CNVT IN

/ADD OE!        :=   RUN
```

```
GAL20V8     7/25/87
FILENAME: 'ut3 dech.pld'  Upper decoder for Definicon interface.
AUTHOR:     James Viq Sherrill

;
;            This is the upper decoder for the definicon systems board
;       that fits inside an IBM PC-AT/XT. It decodes the address into 2
;       sections:
;                                       IN HEX          16th MEG to 17th MEG
;                                                 |           Lower MEG
;                                                 | |
; Address[31:20] decoded on pin 18, 'HIT H'  010X XXXX
; Address[31:16] decoded on pin 19, 'HIT L'  0104 XXXX
;
;


ADD31                       ;1
ADD30                       ;2
ADD29                       ;3
ADD28                       ;4
ADD27                       ;5
ADD26                       ;6
ADD25                       ;7
ADD24                       ;8
ADD23                       ;9
ADD22                       ;10
ADD21                       ;11
GND                         ;GND
ADD20                       ;13
ADD19                       ;14
ADD18                       ;15
ADD17                       ;16
ADD16                       ;17
HIT H!                      ;18
HIT H                       ;19
HIT L                       ;20
EXTGEN!                     ;21
DEL D                       ;22
AS!                         ;23
PWR                         ;PIN 24 IS VCC

;
;Must have a 2 line break between pin descriptions and equations.
;

/HIT H          = /ADD31 * /ADD30 * /ADD29 * /ADD28      ;010X . XXXX
               * /ADD27 * /ADD26 * /ADD25 *  ADD24
               * /ADD23 * /ADD22 * /ADD21 * /ADD20 * /AS!

/HIT L          = /ADD31 * /ADD30 * /ADD29 * /ADD28      ;0104 . XXXX
               * /ADD27 * /ADD26 * /ADD25 *  ADD24
               * /ADD23 * /ADD22 * /ADD21 * /ADD20
               * /ADD19 *  ADD18 * /ADD17 * /ADD16 * /AS!

/HIT H!        = /ADD31 * /ADD30 * /ADD29 * /ADD28      ;010X . XXXX
               * /ADD27 * /ADD26 * /ADD25 *  ADD24
               * /ADD23 * /ADD22 * /ADD21 * /ADD20 * /AS!
```

```
/EXTGEN!          = /ADD31 * /ADD30 * /ADD29 * /ADD28      ;010X , XXXX
                  * /ADD27 * /ADD26 * /ADD25 *  ADD24
                  * /ADD23 * /ADD22 * /ADD21 * /ADD20 * /AS! * DEL D
```

DESCRIPTION

The HIT H is HIT HIGH. This is used to decode any where in the active 1 MEG that the decode logic will be using. The address decoded is set 010X, XXXX in HEX, or 0000 0001 0000 XXXX XXXX XXXX XXXX XXXX in binary. The HIT L bit is used to decode the address a little further. This is from 0100,XXXX in HEX or 0000 0001 0000 0000 XXXX XXXX XXXX XXXX in binary.

```
GAL20V8   7/25/87
FILENAME: 'ut3 decl.pld'   Lower decoder for Definicon interface.
AUTHOR:   James Viq Sherrill

;
;           This is the upper decoder for the definicon systems board
;       that fits inside an IBM PC-AT/XT. It decodes the address into 2
;       sections:
;                                                        BINARY
;                                                           |
;                                                           |
; Address[15:6] decoded on pin 18, 'HIT H'  0000 0000 0XX XXXX;
;


ADD15                    ;1   A[15]
ADD14                    ;2   A[14]
ADD13                    ;3   A[13]
ADD12                    ;4   A[12]
ADD11                    ;5   A[11]
ADD10                    ;6   A[10]
ADD9                     ;7   A[9]
ADD8                     ;8   A[8]
ADD7                     ;9   A[7]
ADD6                     ;10  A[6]
NC0                      ;11
GND                      ;GND
NC1                      ;13
NC2                      ;14
NC3                      ;15
NC4                      ;16
NC5                      ;17
NC6                      ;18
HIT!                     ;19
NC7                      ;20
NC8                      ;21
NC9                      ;22
AS!                      ;23
PWR                      ;PIN 24 IS VCC

;
;Must have a 2 line break between pin descriptions and equations.
;

/HIT!          = /ADD15 * /ADD14 * /ADD13 * /ADD12     ;0000 0000 00XX XXXX
               * /ADD11 * /ADD10 * /ADD9  * /ADD8
               * /ADD7  * /ADD6
               * /AS!

DESCRIPTION

     The Hit output will be active when all the address inputs and AS are
equal to 0. This is not a clocked machine.
```

```
GAL16V8   7/21/87
FILENAME: 'ut3_dsak.pld'        . DSACK* timing generator.
AUTHOR:   James Vig Sherrill

;
;       This will generate the 2 DSACK signals back to the processor. It
;   also contains a watch dog timer to prevent any hang up in the system
;   if an incorrect memory position was decoded. The watchdog will take
;   place only after the HIT line has been activated, and 32 CPU clock
;   cycles have gone by.
;

CLK                          ;1
GP_DA!                       ;2
NC10                         ;3
PT_DA!                       ;4
NC11                         ;5
HIT!                         ;6
RW                           ;7
AS!                          ;8
NC2                          ;9
GND                          ;10  GND
OE                           ;11
CNTO                         ;12
CNT1                         ;13
DSACK1!                      ;14
RD_WR                        ;15
OE!                          ;16
CNT2                         ;17
CNT3                         ;18
CNT4                         ;19
PWR                          ;20 is VCC

;
;Must have a 2 line break between pin descriptions and equations.
;

CNTO          := /HIT! * /CNTO

CNT1          := /HIT! *  CNTO * /CNT1
               + /HIT! * /CNTO *  CNT1

CNT2          := /HIT! * /CNT2 *  CNT1 *  CNTO
               + /HIT! *  CNT2 * /CNT1
               + /HIT! *  CNT2 * /CNTO

CNT3          := /HIT! * /CNT3 *  CNT2 *  CNT1 *  CNTO
               + /HIT! *  CNT3 * /CNT2
               + /HIT! *  CNT3 * /CNT1
               + /HIT! *  CNT3 * /CNTO

CNT4          := /HIT! * /CNT4 *  CNT3 *  CNT2 *  CNT1 *  CNTO
               + /HIT! *  CNT4 * /CNT3
               + /HIT! *  CNT4 * /CNT2
               + /HIT! *  CNT4 * /CNT1
               + /HIT! *  CNT4 * /CNTO
```

```
/DSACK1!      := /HIT! *  CNT2 *          /GP DA! * PT DA!  ;Wait 4 cycles fast
               + /HIT! *  CNT4 *  CNT3 * /PT DA!            ;Wait 28 cycles slo
w

/RD_WR        := /HIT! * /RW * /CNT2 *  CNT1 * /GP DA! *  PT DA!
               + /HIT! * /RW * /CNT2 *  CNTO * /GP DA! *  PT DA!
               + /HIT! * /RW * /CNT4 *  CNT3 * /PT DA!
               + /HIT! * /RW * /CNT4 *  CNT2 * /PT DA!
               + /HIT! * /RW * /CNT4 *  CNT1 * /PT DA!
               + /HIT! * /RW * /CNT4 *  CNTO * /PT DA!
               + /HIT! * /RW *  CNT4 * /CNT3 * /PT DA!

/DE!           =  /HIT! *  CNT1 * CNTO
               + /HIT! * /DE!
 DE!.OE        =  PWR
```

DESCRIPTION

The CNTO - CNT3 are the counter lines. When HIT! is low, these lines will count CPU cycles. When HIT! is high, they are reset to O.

```
*  If the board has been HIT, and neither a PT DA! or a GP DA! respond, then
*  the watch dog timer will generate DSACK1!  after 32  CPU cycles.
*  If a  PT DA or a GP DA has happened, then DSACK! will be generated after 4
*  CPU cycles.
*
*  The PT_DA and GP_DA have been removed.
```

DE! is the data enable for the Bidirectional latches that are on the CPU DATA lines. DE! Becomes active 1 CPU cycle after HIT! becomes active. It stays active as long as HIT! is active.

RD/WR! is the retimed Read/Write line. This signal is based on the RW line generated from the CPU. If RW is high (CPU reading from external RAM) then RD/WR! line will be high. If RW is low (CPU Writing to external RAM) then RD/WR! will go low when HIT! goes low and stay low for 3 CPU cycles. It will go high 1 CPU cycle BEFORE DSACK is generated.

AS RST! is the AS reset line. It is used to reset the FF after AS set it. It is gated with the CPU AS! and DSACK. This is done to make sure that DSACK is asserted long enough (Once DSACK is asserted, it stays asserted until AS! goes inactive.

```
GAL20V8    7/28/87
FILENAME:  UT3_GPAD.PLD       This is the Gereral Purpose D/A Decoder
AUTHOR:    James Vig Sherrill

;
;
;           This gal will decode the DAdress lines and generate the proper
;       select signals for the D/A converter. It will generate all the
;       CS*, RD/WR, and OE lines for the static RAM. They will be
;       tri-stated when not active. The inpu
s are DAddress[19:15], the HIT!
;       signal from the DAdress decoder, INUSE! signal from the D/A logic,
;       RD/WR! signal from the 68020, and the CPU CLOCK.
;
;


NCCLK                   ;1
A19                     ;2
A18                     ;3
A17                     ;4
A16                     ;5
A15                     ;6
HIT!                    ;7
IN_USE!                 ;8
RW                      ;9
DS!                     ;10
NC1                     ;11
GND                     ;12  GROUND
OE                      ;13
NC2                     ;14
CS3!                    ;15
CS2!                    ;16
CS1!                    ;17
CS0!                    ;18
RAM_OE!                 ;19
RD_WR!                  ;20
DE!                     ;21
NC3                     ;22
NC4                     ;23
PWR                     ;24  POWER

;
;Must have a 2 line break between pin descriptions and equations.
;

/CS0!       = /HIT! *   A19 * /A18 * /A17 * /A16     ; 1000 XXXX   8X
/CS0!.OE    = /HIT! *   A19 * /A18 * /A17 * /A16     ; 1000 XXXX   8X

/CS1!       = /HIT! *   A19 * /A18 * /A17 *  A16     ; 1001 XXXX   9X
/CS1!.OE    = /HIT! *   A19 * /A18 * /A17 *  A16     ; 1001 XXXX   9X

/CS2!       = /HIT! *   A19 * /A18 *  A17 * /A16     ; 1010 XXXX   AX
/CS2!.OE    = /HIT! *   A19 * /A18 *  A17 * /A16     ; 1010 XXXX   AX

/CS3!       = /HIT! *   A19 * /A18 *  A17 *  A16     ; 1011 XXXX   BX
/CS3!.OE    = /HIT! *   A19 * /A18 *  A17 *  A16     ; 1011 XXXX   BX

/RAM_OE!    = /HIT! *   A19 * /A18                   ; 10XX XXXX /RW HIT!
```

```
/RAM OE!.OE    = /HIT! *   A19 * /A18              ; 10XX XXXX /RW HIT!

/RD_WR!        = /HIT! *   A19 * /A18 * /RW        ; 10XX XXXX
/RD_WR!.OE     = /HIT! *   A19 * /A18 * /RW        ; 10XX XXXX

/DE!           = /HIT! *   A19                     ; 1XXX XXXX
/DE!.OE        = /HIT! *   A19                     ; 1XXX XXXX
```

DESCRIPTION

This is the decoder set for the D/A section. The CS* lines are mapped a
s
shown below:

```
    0000 0001 0000 0000    0000 0000 0000 0000
                                               > CS0!  0100 0000 to 0100 FFF
F
    0000 0001 0000 0000    0111 1111 1111 1111
    0000 0001 0000 0000    1000 0000 0000 0000
                                               > CS1!  0101 0000 to 0101 FFF
F
    0000 0001 0000 0000    1111 1111 1111 1111
    0000 0001 0000 0001    0000 0000 0000 0000
                                               > CS2!  0102 0000 to 0102 FFF
F
    0000 0001 0000 0001    0111 1111 1111 1111
    0000 0001 0000 0001    1000 0000 0000 0000
                                               > CS3!  0103 0000 to 0103 FFF
F
    0000 0001 0000 0001    1111 1111 1111 1111
```

RAM OE! is the RAM output Enable, this is active low and decoded only when reading from the area 0100 0000 to 0103 FFFF (256k).

RD_WR is the RAM Read/Write line. This is low only when writing to the area 0100 0000 to 0103 FFFF (256k).

DE! is the Data Enable signal that tells the DSACK GAL that data an I/O is taking place in the decoded area. It is active low and only in the area 0100 0000 to 0103 FFFF (256k).

```
GAL20V8   7/28/87
FILENAME: UT3 PTAD.PLD      This is the A/D general I/O port.
AUTHOR:   James Vig Sherrill

;
;        This gal decodes the low level control memory positions for the D/A
;        converter. The decoding is given at the end of the document.
;


NO_CLK                      ;1
A5                          ;2
A4                          ;3
A3                          ;4
A2                          ;5
A1                          ;6
HIT_H!                      ;7
HIT L!                      ;8
RW                          ;9
NOINO                       ;10
NOIN1                       ;11
GND                         ;12   GROUND
NO_OE                       ;13
DS!                         ;14
AD_LOAD                     ;15
AD START                    ;16
AD_OUT1                     ;17
AD OUT1!                    ;18
AD OUT3                     ;19
AD RDWR!                    ;20
AD_READ                     ;21
AD DE!                      ;22
NC4                         ;23
PWR                         ;24   POWER

;
;Must have a 2 line break between pin descriptions and equations.
;

AD_START       =  /HIT H! * /HIT L! *  A5 * /A4 * /A3 * /A2 * /A1 * /RW
AD START.OE    =  PWR
AD LOAD        =  /HIT H! * /HIT L! *  A5 * /A4 * /A3 * /A2 *  A1 * /RW
AD LOAD.OE     =  PWR
AD READ        =  /HIT H! * /HIT L! *  A5 * /A4 * /A3 *  A2 * /A1 *  RW
AD READ.OE     =  PWR
AD OUT1        =  /HIT_H! * /HIT_L! *  A5 * /A4 * /A3 *  A2 *  A1 * /RW
AD OUT1.OE     =  PWR
/AD_OUT1!      =  /HIT H! * /HIT_L! *  A5 * /A4 * /A3 *  A2 *  A1 * /RW
/AD OUT1!.OE   =  PWR
AD OUT3        =  /HIT_H! * /HIT_L! *  A5 * /A4 *  A3 * /A2 * /A1 * /RW
AD OUT3.OE     =  PWR


/AD RDWR!      =  /HIT_H! * /HIT L! *  A5 * /A4 * /RW
 AD RDWR!.OE   =  /HIT_H! * /HIT L! *  A5 * /A4   /RW

/AD_DE!        =  /HIT H! * /HIT L! *  A5 * /A4
 AD DE!.OE     =  /HIT H! * /HIT_L! *  A5 * /A4
```

DESCRIPTION

Signal	RD/WR	Address decode
AD_START	WR only	0104 0020
AD_LOAD	WR only	0104 0022
AD_READ	RD only	0104 0024
AD_OUT1	WR only	0104 0026
AD_OUT2	WR only	0104 0028
AD_OUT3	WR only	0104 002A

```
GAL20V8   7/28/87
FILENAME: UT3 PTDA.PLD      This is the D/A general I/O port.
AUTHOR:   James Vig Sherrill

;
;       This gal decodes the low level control memory positions for the D/A
;       converter. The decoding is given at the end of the document.
;


NO_CLK                  ;1
A5                      ;2
A4                      ;3
A3                      ;4
A2                      ;5
A1                      ;6
HIT_H!                  ;7
HIT_L!                  ;8
RW                      ;9
NOINO                   ;10
NOIN1                   ;11
GND                     ;12  GROUND
NO_OE                   ;13
DS!                     ;14
DA_DE!                  ;15
DA_OUT                  ;16
DA_REG                  ;17
DA_START                ;18
DA_AL                   ;19
DA_LL                   ;20
DA_PT_WR                ;21
DA_DSAK!                ;22
NC4                     ;23
PWR                     ;24  POWER

;
;Must have a 2 line break between pin descriptions and equations.
;

DA_START      =  /HIT_H! * /HIT_L! * /A5 * /A4 * /A3 * /A2 * /A1 * /RW
DA_START.OE   =  PWR

DA_REG        =  /HIT_H! * /HIT_L! * /A5 * /A4 * /A3 * /A2 *  A1 * /RW
DA_REG.OE     =  PWR

DA_AL         =  /HIT_H! * /HIT_L! * /A5 * /A4 * /A3 *  A2 * /A1 * /RW
DA_AL.OE      =  PWR

DA_LL         =  /HIT_H! * /HIT_L! * /A5 * /A4 * /A3 *  A2 *  A1 * /RW
DA_LL.OE      =  PWR

DA_OUT        =  /HIT_H! * /HIT_L! * /A5 * /A4 *  A3 * /A2 *  A1 * /RW
DA_OUT.OE     =  PWR

DA_PT_WR      =  /HIT_H! * /HIT_L! * /A5 * /A4 *  A3 *  A2 * /A1 * /RW
DA_PT_WR.OE   =  PWR

/DA_DSAK!     =  /HIT_H! * /HIT_L!
```

```
/DA DSAK!.OE   =  PWR

/DA_DE!        =  /HIT H! * /HIT L! * /A5 *  /A4 * /A3 * /A2 *  A1   ;DA REG
               +  /HIT H! * /HIT L! * /A5 *  /A4 * /A3 *  A2 * /A1   ;DA AL
               +  /HIT H! * /HIT L! * /A5 *  /A4 * /A3 *  A2 *  A1   ;DA LL
               +  /HIT H! * /HIT_L! * /A5 *  /A4 *  A3 * /A2 * /A1   ;DA GP
               +  /HIT H! * /HIT L! * /A5 *  /A4 *  A3 * /A2 *  A1   ;DA CNVT
               +  /HIT H! * /HIT_L! * /A5 *  /A4 *  A3 *  A2 * /A1   ;DA RD


/DA DE!.OE     =  /HIT H! * /HIT L! * /A5
```

DESCRIPTION

Signal	RD/WR	Address decode
DA START	WR only	0104 0000
DA_REG	WR only	0104 0002
DA AL	WR only	0104 0004
DA_LL	WR only	0104 0006
DA OUT	WR only	0104 000A
DA_DA_PT	WR only	0104 000C

DA DE   Output enable for data transfers

/DA DSAK!   This signal is used to signal back to the DSACK generator that an access to the range 0104 000X has occurred.

-*-*-*- APPENDIX B -*-*-*-

Source code listings for freq gen.c and associated code, plot1.c, g1.c, and graph.c, which together comprise the frequency generation and analysis package.

```
#include "stdio.h"
#include "math.h"

/************************************************************************

        Definitions-

*************************************************************************/


                    /* Colors for IBM pc ROM-BIOS calls */

#define black     0x00
#define blue      0x01
#define green     0x02          /* COLORS are 4 bits long for mode 3 */
#define cyan      0x03
#define red       0x04
#define purple    0x05
#define brown     0x06
#define white     0x07

#define lblack    0x08
#define lblue     0x09
#define lgreen    0x0a          /* These colors, when used as background, */
#define lcyan     0x0b          /* make the foreground blink.
*/
#define lred      0x0c
#define lpurple   0x0d
#define yellow    0x0e
#define lwhite    0x0F

#define hl_color_fg       yellow
#define hl_color_bg       black
#define uhl_color_fg      white      /* Hilighted and Non hilighted colors */
#define uhl_color_bg      blue

#define m_bar_color_fg    yellow
#define m_bar_color_bg    blue
#define fld_color_fg      white
#define fld_color_bg      blue
#define opt_color_fg      white
#define opt_color_bg      blue


#define fr_color_fg        yellow
#define fr_color_bg        blue
#define fr_bar_color_fg    yellow
#define fr_bar_color_bg    blue
#define fr_hl_color_fg     yellow
#define fr_hl_color_bg     black
#define fr_uhl_color_fg    lwhite
#define fr_uhl_color_bg    blue          /* frequency editor stuff */
#define fr_get_color_fg    lred
#define fr_get_color_bg    blue
#define fr_mess_color_fg   lgreen
#define fr_mess_color_bg   blue
```

```
#define fr_box_leng       77
#define fr_fld_leng       9
#define fr_page           0
#define fr_mess_row       3
#define fr_mess_col       3
#define fr_box_row        1
#define fr_box_col        1
#define frgs_col          30
#define frgs_row          3



#define quit_col        0
#define quit_row        1          /* Menu position indices */
#define go_col          0
#define go_row          0
#define first_col       1
#define first_row       0

                     /* Hex versions of Box characters */

#define tl       0xda
#define tr       0xbf
#define bl       0xc0
#define br       0xd9
#define hzt      0xc4
#define vrt      0xb4      /* single width box characters */
#define lm       0xc3
#define rm       0xb3
#define tm       0xc2
#define bm       0xc1
#define center   0xc5

#define tl_d     0xc9
#define tr_d     0xbb
#define bl_d     0xc8
#define br_d     0xbc
#define hzt_d    0xcd      /* Double width box characters */
#define vrt_d    0xba
#define lm_d     0xcc
#define rm_d     0xb9
#define tm_d     0xcb
#define bm_d     0xca
#define center_d 0xce


#define blank     0x20

#define max_menu 7
#define max_row  7
#define max_col  2                /* dimensions for action vectors */
#define max_key  6
#define max_func_dimen     5

#define right_key          0x4d00
#define left_key           0x4b00
#define up_key             0x4800          /* keyboard key values to int86 */
```

```
#define down_key          0x5000
#define end_key           0x4f00
#define return_key        0x1c0d
#define plus_key          0x4e2b
#define minus_key         0x4a2d

#define mess_row          3
#define mess_col          3

#define stat_row          18
#define stat_col          46

#define act_row           3
#define act_col           43

#define sel_row           3
#define sel_col           58                 /* row and col positions of graphics
*/

#define que_row           18
#define que_col           3

#define inf_row           10
#define inf_col           3

#define box_row           4
#define box_col           40

#define sf1_row           box_row+2
#define sf1_col           box_col+2

#define sf2_row           box_row+2
#define sf2_col           box_col+17

#define f1_leng            11
#define f2_leng            25

#define mess_und_leng      18
#define mess_fld_leng      34

#define que_und_leng       18
#define que_fld_leng       38

#define stat_und_leng      16
#define stat_fld_leng      34

#define inf_fld_leng       36


int page;

char in_file[30];         /* file i/o variables */
char out_file[30];
char bigbuf[4096];
FILE *io_ptr;


/****************************************************************************
```

```
        Definicon register descriptions for PC interfacing

*****************************************************************************/

struct REGS {
        unsigned short ax;
        unsigned short flags;
        unsigned short bx;
        unsigned short cx;
        unsigned short dx;
        unsigned short si;
        unsigned short di;
        unsigned short ds;
        unsigned short es;

}  inregs;



/****************************************************************************

   Positions are structures containing the values which specify a particular
place in the system.  Often these values will be used as indices into other
structures in order to find out a relevant action for that position.

****************************************************************************/

struct position
{
        int menu;
        int row;
        int col;

}  pos;

/****************************************************************************

   Selection matrices hold a picture of what has been selected for any
given menu.

****************************************************************************/

struct selection_matrix
{
        int menu;
        int row [max_row];
        int col [max_col];

} current_sm;

/****************************************************************************

    Action Vectors consist of a count of the number of functions (or
actions) to carry out, along with an array of pointers to these functions.
```

```
    A multi-dimensional array of pointers to action vectors allows the
creation of "action chains" for each position/key combination.  The indices
for these dimensions are the members of a position structure and a key value.

*************************************************************************/

struct action_vector
{
        int count;
        void ( *func[max_func_dimen] ) ();

};

struct position_info
{
        int length;
        int abs_row;
        int abs_col;

        struct action_vector   act_vect [max_key];
};


/*************************************************************************

        Memory map of Defincicon/Ram interface

*************************************************************************
/


#define ca_ram              0x01000000

#define da_start            0x01040000
#define ca_req              0x01040002              /* D to A section */
#define da_ai               0x01040004
#define da_ll               0x01040006
#define da_out              0x0104000a

#define ad_ram              0x01080000

#define ad_start            0x01040020
#define ad_load             0x01040022              /* A to D section */
#define ad_read             0x01040024
#define ad_out1             0x01040026

#define go_val              0


        /* global vars */

#define pi                 3.14159265
#define max_int_size       32767
#define points_per_sec     1.0e6


/***********************************************************************
```

```
        parm set to completely specify a run

*****************************************************************************/

        /* transmission/reception control parms */

float total_time;
float ramp_time;
float decay_time;

float record_time;
float delay_time;

int loop_start;
int loop_end;
int loop_count;


        /* Frequencies and Points */

long num_of_points;
int num_of_freqs;
int cur_freq_num;

float freq[100];
float point[70000];


        /* Swept frequency parameters */

float freq_per_time;
float fp_s_freq;
float fp_e_freq;
float s_freq;
float e_freq;
float mix_lag;

unsigned long mix_lag_cnt;
unsigned long mix_lag_cnt2;


        /* misc  flags */

int xmit_mode;
int test_dac_out;


/***************************/
/*** main.c  boundpoint   ***/
/***************************/

/* (NOTE: the boundpoint message is to relate this section to a
    corresponding section of code written for microsoft operation */


/*****************************************************************************
```

```
                         MAIN

        Prints first screen.  Zeros ram.
        Does appropriate action as long as keys are hit.

*****************************************************************************/

main ()
{
        pr_common();
        pr_menu0();

        num_of_freqs=0;
        mix_lag = 0.0;

        while (1)
        {
                take_action( keynum ( getkey () ));
        }

}



/*****************************/
/*** board2.c  boundpoint ***/
/*****************************/


/***************************************************************************

        Zeros the A to D ram (128k of 2 bytes)

***************************************************************************/

zero_ad()
{
short *p;
int i;
        p= ad_ram;
        for (i=0; i < 0x00020000; i++) *p++ = 0.0;

}


/***************************************************************************

        Zeros the D to A ram (128k of 2 bytes)

***************************************************************************/

zero_da()
{
short *p;
int i;
        p = da_ram;
        for (i=0; i < 0x00020000; i++) *p++ = 0.0;
```

```
}


/**********************************************************************

        Starts the D to A rolling out values

**********************************************************************/

play()
{
short *p;
        /* let it roll, don't wait or stop */

        p = da_start;
        *p = go_val;

}


/**********************************************************************

        Find values for loop control and point placement.

        Called after total, ramp, and decay times have been entered.

**********************************************************************/

calc_loop_vals()

{
float centre;
float remain;
int loops;

        /* figure out number of 10 msec loops needed */

        centre = total_time - (decay_time + ramp_time);
        loops  = centre/10;
        remain = centre - ( 10 * loops );


        if ( loops == 0 )          /* no looping to be done */
         {
                loop_end = 0;
                loop_start = 0;

                num_of_points = total_time / 1000.0 * points_per_sec;

         }
        else                       /* put loop in proper place and number */
         {

                loop_start = (ramp_time + remain) * 1000;
                loop_end = loop_start + 10000;
```

```
		num_of_points = (ramp_time + remain + 10 + decay_time) * 1000
;

	}

	loop_count = 8192 - loops;	/* 0x2000 - loop_count */
}



/**********************************************************************

	Find values for sweep control and point placement.

	Called after total, ramp, and decay times; and starting and ending
	frequencies have been entered.

**********************************************************************/

calc_sweep_vals()

{
float centre;
unsigned long delay, t_5khz;

	/* length of center full power region */

	centre = total_time - (ramp_time + decay_time);


	/* NOTE: Full Power end freqeunecy is effectively fp_e_freq + 5 khz */
	/* freq per time is in  Hz pur point  (or KHz per msec) */

	freq_per_time = (fp_e_freq + 5 - fp_s_freq) / centre;


	/* find starting and ending frequencies */

	s_freq = fp_s_freq - ( freq_per_time * ramp_time );
	if (s_freq < 0)
		printf ("Ramp time too long for starting freq and burst time.
");

	e_freq = fp_e_freq + 5 + ( freq_per_time * decay_time ) ;


	/* find out values for address counters (mix lag counts) */

	cmlcs();


	/* set number of points to do and specify no looping */

	num_of_points = total_time / 1000.0 * points_per_sec;
	loop_start = 0;
	loop_end = 0;
```

```
}



/*************************************************************************

        Load address 1

*************************************************************************/

load_addr1(addr)
unsigned long addr;
{
short *p;

        p = da_reg;
        *p = addr;          /* address in address register */

        p = da_al;
        *p = go_val;        /* hit the address latch to move address along path
*/
}

/*************************************************************************

        Load address 2

*************************************************************************/

load_addr2(addr)
unsigned long addr;
{
short *p;

        p = da_reg;
        *p = addr;          /* address in address register */

        p = da_al;
        *p = go_val;        /* hit the address latch to move address along path
*/
}



/*************************************************************************

        Load the address to begin the loop.

*************************************************************************/

load_loop_start()
{
short *p;

        p = da_reg;
        *p = loop_start;  /* address in address register */
```

```
}

/****************************************************************************

        Load number of times to loop.

****************************************************************************/

load_loop_count()
{
short *p;

        p = da_l1;
        *p = loop_count; /* loop count in loop counter */
}


/****************************************************************************

        Load the array of points to the ram starting at 0, going for
        number of points. Last point is a stop value. Loop values set
        if needed.

****************************************************************************/

load_points()

{
unsigned long i;
short *p;

        p = da_ram;

        if (loop_end != 0)   /* skip this section if not looping i.e. [illegible]
end */
          {

                for (i=0; i < loop_end - 1 ; i++)
                 {                       /* invert MSB , zero bottom four bits *
/
                        *p++ = ((short) point[i] ^ 0x8000) & 0xfff0;
                 }

                /* load loop end */
                                          /* invert MSB , set loop bit */
                *p++ = (((short) point[i] ^ 0x8000) & 0xfff0) | 0x0002;
          }


        for (i = loop_end ; i < num_of_points - 1 ; i++)
          {
                    /* invert MSB , zero bottom four bits */
                *p++ = ((short) point[i] ^ 0x8000) & 0xfff0;
          }
```

```
        if (xmit_mode == 1) /* if sweep mode .... */
         {
                /* zero out any remaining points  */

                for (i = 0 ; i < (0x1ffff - num_of_points) ; i++)
                 {
                        *p++ = 0x8000; /* invert MSB, send zero */
                 }
         }

        /* load stop */

        *p++ = 0x8001; /* invert MSB, set stop bit */

}

/****************************************************************************

   generate complex wave based upon supplied frequencies in

        freq [num_of_freqs].

*****************************************************************************/

wave_gen()

{
float  scale_factor, temp, incr, two_pi, max_val;
int        n;
unsigned long i;

        two_pi = 2.0 * pi;
        max_val = 0;

        /* start with clean slate */
        for (i=0; i < num_of_points ; i++) point[i] = 0.0;


        /* do for all frequencies */

        for (n=0 ; n < num_of_freqs ; n++)
         {

                incr = two_pi * freq[n] * 1000.0 / points_per_sec;
                temp = 0;

                /* do for all points */

                for (i=0; i < num_of_points ; i++)
                 {
                        temp += incr;
                        if (temp > two_pi) temp -= two_pi;

                        point[i] += sin(temp);

                        if (point[i] > max_val) max_val = point[i];
```

```
                }
          }

        /* scale for max value found */

        scale_factor =  -1.0 * max_int_size / max_val;

        for (i=0; i < num_of_points; i++)
         {
                point[i] *= scale_factor;          /* -32k to 32 k */
         }

        ramp_up();
        decay();

        load_points();
}


/*****************************************************************************

          generate swept frequency wave based upon supplied frequencies and
        time parameters.

*****************************************************************************/

wave_gen2()

{
float   scale_factor, temp, i_xs_inc2, inc1, inc2, two_pi, max_val;
int     n;
unsigned long i;

        two_pi = 2.0 * pi;
        max_val = 0;

        /* start with clean slate */
        for (i=0; i < num_of_points ; i++) point[i] = 0.0;

        inc1 = two_pi * s_freq * 1000.0 / points_per_sec; /* s_freq is KHz
*/
        inc2 = two_pi * freq_per_time / points_per_sec;    /* freq_per_time is
Hz */

        temp = 0;
        i_xs_inc2 = 0;


        /* Fill point array with frequency sweep */

        for (i=0; i < num_of_points ; i++)
         {
                temp += ( inc1 + i_xs_inc2 );

                if (temp > two_pi) temp -= two_pi;    /* keep argument small *
/
```

```
                point[i] += sin(temp);

                i_xs_inc2 += inc2;                          /* incr = inc1 + i(inc
2) */

                if (point[i] > max_val) max_val = point[i];
            }


        /* scale for max value found */

        scale_factor =  -1.0 * max_int_size / max_val;

        for (i=0; i < num_of_points; i++)
            {
                point[i] *= scale_factor;         /* -32k to 32 k */
            }

        ramp_up();
        decay();

        load_points();
}



/*******************************************************************************

        Ramp up linearly from 0 address for specified # of milliseconds.
        the ramp_time.

*******************************************************************************/

 amp_up()

{
int points_to_ramp, i;
float factor, incr;

        points_to_ramp = ramp_time / 1000.0 * points_per_sec;
        incr = 1.0 / points_to_ramp;
        factor = 0;

        for (i=0; i < points_to_ramp ; i++)
            {
                factor += incr;
                point[i] *= factor;
            }
}



/*******************************************************************************

        Decay linearly for specified # of miliseconds, the decay time,
        before the end.
```

```
***************************************************************************/

decay()

{
int points_to_decay;
unsigned long i;
float factor, incr;

        points_to_decay = decay_time / 1000.0 * points_per_sec;

        incr = 1.0 / points_to_decay;
        factor = 1.0;


        for (i = num_of_points - points_to_decay; i < num_of_points; i++)
         {
                point[i] *= factor;
                factor -= incr;
         }
}




/**************************/
/*** board3.c  boundpoint ***/
/**************************/


/**************************************************************************

          Put a stop bit at the proper number of points past the start
        of the A to D ram.

***************************************************************************/

load_record_time()
{
int a;
short *p;

        p = ad_ram;
        a =  record_time * 1000;
        *(p+a) = 0x0001;

}


/**************************************************************************

           load the delay counter with 0xffff minus the number of micro
        seconds to delay.

**************************************************************************/
```

```
load_delay_time()
{
short *p;

        p = ad_load;
        *p = 0xffff - ((int) delay_time * 1000);
}



/***************************/
/*** clear.c  boundpoint  ***/
/***************************/

/***********************************************************************

          These routines clear conceptually defined sections of the
        interface screen.

***********************************************************************/

clear_select ()
{
int i;

        /* clear menu selection fields */

        for (i=box_row+1; i<box_row+12; i++)
        {       corchar(i, box_col+1, opt_color_fg, opt_color_bg, 0, f1_leng,
blank);
                corchar(i, box_col+13, opt_color_fg, opt_color_bg, 0, f2_leng
, blank);
        }
}

clear_status ()
{
int i;

        /* clear status section */

        for (i=stat_row+2; i<stat_row+5; i++)
        corchar(i, stat_col, black, black, 0, stat_fld_leng, blank);
}

clear_messages ()
{
int i;

        /* clear message section */

        for (i=mess_row+2; i<mess_row+5; i++)
        corchar(i, mess_col, white, black, 0, mess_fld_leng, blank);
}
```

```
clear_questions ()
{
int i;

        /* clear questions section */

        for (i=que_row+2; i<que_row+5; i++)
        cprchar(i, que_col-2, white, black, 0, que_fld_leng+2, blank);
}


clear_info ()
{
int i;

        /* clear info section */

        for (i=inf_row; i<inf_row+8; i++)
        cprchar(i, inf_col, white, black, 0, inf_fld_leng, blank);
}


clear_screen ()
{
        /* clear screen */

        cprchar(1, 1, white, black, 0, 1920, blank);
}


blank_fields ()
{
        clear_select();
        clear_status();
        clear_questions();
        clear_info();
}




/***************************/
/*** efreqs.c  boundpoint ***/
/***************************/

/*************************************************************************************

          This section deals with the frequency editor (which allows the
        entry of frequencies desired)
```

```
*****************************************************************************
/

/****************************************************************************

        Print the background screen for the editor

*****************************************************************************
/

make_fr_backgnd()
{
int i;
        /* blank screen */

        cprchar(fr_box_row, fr_box_col, fr_color_fg, fr_color_bg,
                fr_page, 1920, blank);


        /* print corners */

        cprchar(fr_box_row, fr_box_col, fr_bar_color_fg, fr_bar_color_bg,
                fr_page, 1, tl_d);
        cprchar(fr_box_row, fr_box_col+78, fr_bar_color_fg, fr_bar_color_bg,
                fr_page, 1, tr_d);
        cprchar(fr_box_row+22, fr_box_col, fr_bar_color_fg, fr_bar_color_bg,
                fr_page, 1, bl_d);
        cprchar(fr_box_row+22, fr_box_col+78, fr_bar_color_fg, fr_bar_color_b
g,

                fr_page, 1, br_d);


        /* horizontal bars */

        cprchar(fr_box_row, fr_box_col+1, fr_bar_color_fg, fr_bar_color_bg,
                fr_page, fr_box_leng, hzt_d );
        cprchar(fr_box_row+22, fr_box_col+1, fr_bar_color_fg, fr_bar_color_bg

                fr_page, fr_box_leng, hzt_d );


        /* vertical bars */

        for (i= fr_box_row+1; i < fr_box_row+22; i++)
         {
                cprchar(i, fr_box_col, fr_bar_color_fg, fr_bar_color_bg,
                        fr_page, 1, vrt_d );
                cprchar(i, fr_box_col+78, fr_bar_color_fg, fr_bar_color_bg,
                        fr_page, 1, vrt_d );
         }
}


/****************************************************************************

        Set mode (sweep or discrete) and get the frequencies of interest.
```

```
*****************************************************************************
/

get_freqs()

{
        make_fr_backgnd();

        get xmit_mode();

        if (xmit_mode == 1)
                get_sw_freqs();
        else
                get_dis_freqs();

}


/****************************************************************************

        Get and set the transmission mode.

*****************************************************************************
/

get_xmit_mode()

{
int i, temp;

        /* print messages telling how to set ... */

        cprintf(fr_mess_row+2, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
                fr_page, " Choose mode of frequency transmission.");
        cprintf(fr_mess_row+3, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
                fr_page, "     1 - Swept frequency     ");
        cprintf(fr_mess_row+4, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
                fr_page, "     0 - Discrete frequency ");


        /* and get the mode desired */

        cprintf(fr_mess_row+7, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
                fr_page, " -> ");
        scanf("%d", &temp);

        xmit_mode = temp;


        /* clear message space */

        for (i=0; i<13; i++)
                cprchar(fr_mess_row+2+i, fr_mess_col, fr_color_fg, fr_color_b
g,
```

```
                    fr_page, 40, blank);

}


/*************************************************************************

        Get and set the sweep frequencies.

**************************************************************************
/

get_sw_freqs()

{
        get_start_freq();
        get_end_freq();

        clear_screen();
        pr_common();
        pr_menu1();
}


/*************************************************************************

        Get and set the starting frequency of a sweep.

**************************************************************************
/

get_start_freq()

{
int i;
float temp;

        /* print messages telling how to set ... */

        cprintf(fr_mess_row+2, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
                fr_page, " Choose starting frequency.");
        cprintf(fr_mess_row+3, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
                fr_page, "   (Range 30.0 - 100.0 Khz) ");


        /* and get the frequency desired */

        cprintf(fr_mess_row+5, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
                fr_page, " -> ");
        scanf("%f", &temp);

        fp_s_freq = temp;
```

```
        /* clear message space */

        for (i=0; i<13; i++)
                cprchar(fr_mess_row+2+i, fr_mess_col, fr_color_fg, fr_color_b
g,
                        fr_page, 40, blank);

}

/****************************************************************************

        Get and set the ending frequency of a sweep.

****************************************************************************
/

get_end_freq()

{
int i;
float temp;

        /* print messages telling how to set ... */

        cprintf(fr_mess_row+2, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
                fr_page, " Choose ending frequency.");
        cprintf(fr_mess_row+3, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
                fr_page, "    (Range 30.0 - 100.0 Khz) ");


        /* and get the frequency desired */

        cprintf(fr_mess_row+5, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
                fr_page, " -> ");
        scanf("%f", &temp);

        fp_e_freq = temp;


        /* clear message space */

        for (i=0; i<13; i++)
                cprchar(fr_mess_row+2+i, fr_mess_col, fr_color_fg, fr_color_b
g,
                        fr_page, 40, blank);

}

/****************************************************************************

        Get and set discrete frequencies. (up to 100)
```

```
*****************************************************************************
/

get_dis_freqs()

{
float temp;
int i,  gf_go, choice;


        /* print current freqs */

        if (num_of_freqs == 0)
                cprintf(frqs_row, frqs_col, lred, black, fr_page, "1) ");
        else
                for (i = 0 ; i < num_of_freqs ; i++)
                        cprintf( frqs_row + i - ( 20 * ( i / 20 )), frqs_col+
((i/20)*9),
                                fr_uhl_color_fg, fr_uhl_color_bg,
                                fr_page, "%u) %.1f", i+1, freq[i]);


        /* hilight position */

        cur_freq_num = 0;
        fr_hilight();


        /* print labels */

        cprintf(fr_mess_row, fr_mess_col+2, fr_bar_color_fg, fr_bar_color_bg
                fr_page, " MESSAGES ");
        cprchar(fr_mess_row+1, fr_mess_col, fr_bar_color_fg, fr_bar_color_bg,
                fr_page, 12, hzt_d );

/*      cprintf(fr_mess_row, fr_mess_col+35, fr_bar_color_fg, fr_bar_color_bg
,
                fr_page, " FREQUENCIES ");
        cprchar(fr_mess_row+1, fr_mess_col+33, fr_bar_color_fg, fr_bar_color_
bg,
                fr_page, 15, hzt_d );
*/


        /* print instructions */

        cprintf(fr_mess_row+2, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
                fr_page, " ARROWS position bar ");
        cprintf(fr_mess_row+3, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
                fr_page, " RETURN to enter values ");
        cprintf(fr_mess_row+4, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
                fr_page, " END to quit editing ");


        /* process input */

        gf_go = 1;
```

```
        while ( gf_go )
         {

                choice = keynum( getkey() );

                switch (choice)           /* Set current frequency number and
hilight new place */
                 {
                        case 0: /* down_key */

                                if (cur_freq_num < num_of_freqs)
                                 {
                                        fr_unhilight();
                                        cur_freq_num += 1;
                                        fr_hilight();
                                 }
                                break;


                        case 1: /* up_key */

                                if (cur_freq_num > 0)
                                 {
                                        fr_unhilight();
                                        cur_freq_num -= 1;
                                        fr_hilight();
                                 }
                                break;


                        case 2: /* right_key */

                                if (cur_freq_num+20 < num_of_freqs)
                                 {
                                        fr_unhilight();
                                        cur_freq_num += 20;
                                        fr_hilight();
                                 }
                                break;


                        case 3: /* left_key */

                                if (cur_freq_num-20 >= 0)
                                 {
                                        fr_unhilight();
                                        cur_freq_num -= 20;
                                        fr_hilight();
                                 }
                                break;


                        case 4: /* end_key */
                                gf_go = 0;        /* get out */
                                break;

                        case 5: /* return_key */
                                enter_freqs();
```

```
			}

		}

		clear_screen();
		pr_common();
		pr_menu1();
}

/****************************************************************************

		Actually enter the frequency values

*****************************************************************************
/

enter_freqs()

{
int i;
float temp;

		/* clear message space */

		for (i=0; i<13; i++)
			cprchar(fr_mess_row+2+i, fr_mess_col, fr_color_fg, fr_color_b
g,
				fr_page, 27, blank);

		/* print messages telling how to get ... */

		cprintf(fr_mess_row+2, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
			fr_page, " Enter frequencies in Khz.");
		cprintf(fr_mess_row+3, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
			fr_page, "    Range is 30.0 - 100.0 ");
		cprintf(fr_mess_row+4, fr_mess_col, fr_mess_color_fg, fr_mess_color_b
g,
			fr_page, "    Enter 1.0 to quit.    ");


		/* and get the latest frequency */

		cprintf(fr_mess_row+6, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
			fr_page, " Frequency # %u", cur_freq_num+1);
		cprintf(fr_mess_row+7, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
			fr_page, " -> ");
		scanf("%f", &temp);

		cprchar(fr_mess_row+7, fr_mess_col, fr_color_fg, fr_color_bg,
			fr_page, 10, blank);


		i = cur_freq_num;
```

```
        while ((temp != 1.0) && (i<100))    /* do until signal to quit is ente
red */
         {

                if ((temp <= 100.0) && (temp >= 30.0)) /* but only if in rang
e */
                 {
                        freq[i] = temp;
                        cprintf( frqs_row + i - ( 20 * ( i / 20 )), frqs_col+
((i/20)*9),

                                fr_uhl_color_fg, fr_uhl_color_bg,
                                fr_page, "%u) %.1f", i+1, freq[i]);

                        fr_unhilight();
                        cur_freq_num += 1;
                        i += 1;
                        fr_hilight();

                 }

                else    /* out of range message */

                        cprintf(fr_mess_row+10, fr_mess_col+2, yellow, lred,
                                fr_page, " OUT OF RANGE ");


                /* get next frequency entry */

                cprintf(fr_mess_row+6, fr_mess_col, fr_get_color_fg, fr_get_c
olor_bg,
                        fr_page, " Frequency # %u", cur_freq_num+1);
                cprintf(fr_mess_row+7, fr_mess_col, fr_get_color_fg, fr_get_c
olor_bg,
                        fr_page, " -> ");
                scanf("%f", &temp);

                cprchar(fr_mess_row+7, fr_mess_col, fr_color_fg, fr_color_bg,
                        fr_page, 10, blank);
                cprchar(fr_mess_row+10, fr_mess_col, fr_color_fg, fr_color_bg
,
                        fr_page, 16, blank);


         }

        if (i > num_of_freqs)       num_of_freqs = i;

        /* clear message space */

        for (i=0; i<13; i++)
                cprchar(fr_mess_row+2+i, fr_mess_col, fr_color_fg, fr_color_b
g,
                        fr_page, 27, blank);


        /* print instructions */

        cprintf(fr_mess_row+2, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
```

```
                fr_page, " ARROWS position bar ");
        cprintf(fr_mess_row+3, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
                fr_page, " RETURN to enter values ");
        cprintf(fr_mess_row+4, fr_mess_col, fr_get_color_fg, fr_get_color_bg,
                fr_page, " END to quit editing ");

}

/***************************************************************************

        FRequency HILIGHT and UNHILGHT -

                These routines
        hilight and unhilight a field in frequency selection situation.

***************************************************************************/

fr_hilight ()
{
        fr_color_change ( fr_hl_color_bg, fr_hl_color_fg );
}


fr_unhilight ()
{
        fr_color_change ( fr_uhl_color_bg, fr_uhl_color_fg );
}


fr_color_change ( color_bg, color_fg )

int color_bg, color_fg;
{
int   times, row, col, page;


        times = fr_fld_leng;
        row = frqs_row + cur_freq_num - ( 20 * ( cur_freq_num / 20 ));
        col = frqs_col + (( cur_freq_num / 20 ) * 9 );
        page = fr_page;

        while (times-- > 0)                         /* for the length of field
*/
        {
                curset ( row, col++, page);

                inregs.ax = 0x0800;                 /* read the character and
*/
                inregs.bx = page << 8;              /* and it's attribute
*/
                INT86(0x10,&inregs );


                inregs.ax = inregs.ax & 0x00ff; /* write back same char     */
                inregs.ax |= 0x0900;
                inregs.bx = inregs.bx & 0xff00; /* same page */
                inregs.bx |= ((color_bg << 4)
                        + color_fg);        /* different attributes    */
```

```
                inregs.cx = 1;                  /* one copy (no repeats)    *
/
                INT86(0x10,&inregs );
        }
}


/***************************/
/*** ftrans.c  boundpoint ***/
/***************************/

/***************************************************************************
*

        This section deals with file transfers of parms and data.

****************************************************************************
*/


/***************************************************************************
*

        Transfer a file to current parameter set.

****************************************************************************
*/

file_to_parms()

{
unsigned long i;
float temp;

        /* Print messages */

        clear_status();

        cprintf(stat_row+2, stat_col, purple, lgreen, 0, " Transfering from
");
        cprintf(stat_row+3, stat_col, purple, lgreen, 0, "  file to parms ...
");


        if ((io_ptr = fopen(in_file,"r")) == 0 )
         {
                cprintf(inf_row+2, inf_col, yellow, lred, 0, " Can't create i
nput file: ");
                cprintf(inf_row+3, inf_col, yellow, lred, 0, " < %s  >  ",in_
file);

                return;
         }

        setbuf (io_ptr, bigbuf);
```

```
        /* load parms from file into currently active parms */

        fscanf(io_ptr, "%f ", &total_time);
        fscanf(io_ptr, "%f ", &ramp_time);
        fscanf(io_ptr, "%f ", &decay_time);

        fscanf(io_ptr, "%f ", &record_time);
        fscanf(io_ptr, "%f ", &delay_time);

        fscanf(io_ptr, "%u ", &loop_start );
        fscanf(io_ptr, "%u ", &loop_end );
        fscanf(io_ptr, "%u ", &loop_count );

        fscanf(io_ptr, "%u ", &num_of_freqs );
        fscanf(io_ptr, "%u ", &num_of_points );

        for (i=0; i < num_of_freqs; i++ )
         {
                fscanf(io_ptr, "%f ", &temp );
                freq[i] = temp;
         }
        for (i=0; i < num_of_points; i++ )
         {
                fscanf(io_ptr, "%f ", &temp );
                point[i] = temp;
         }

        fclose(io_ptr);
        clear_status();

}

/*****************************************************************************
*

        Transfer current parameter set to a file.

*****************************************************************************
*/

parms_to_file()

{
unsigned long i;

        /* Print messages */

        clear_status();

        cprintf(stat_row+2, stat_col, purple, lgreen, 0, " Transfering from
");
        cprintf(stat_row+3, stat_col, purple, lgreen, 0, "  parms to file ...
");
```

```
        if ((io_ptr = fopen(out_file,"w")) == 0 )
         {
                cprintf(inf_row+2, inf_col, yellow, lred, 0, " Can't create o
utput file: ");
                cprintf(inf_row+3, inf_col, yellow, lred, 0, " < %s  >  ",out
_file);

                return;
         }

        setbuf (io_ptr, biobuf);


        /* load parms from currently active parms into file */

        fprintf(io_ptr, "% .3f", total_time);
        fprintf(io_ptr, "% .3f", ramp_time);
        fprintf(io_ptr, "% .3f", decay_time);

        fprintf(io_ptr, "% .3f", record_time);
        fprintf(io_ptr, "% .3f", delay_time);

        fprintf(io_ptr, " ");

        fprintf(io_ptr, "% u ", loop_start );
        fprintf(io_ptr, "% u ", loop_end );
        fprintf(io_ptr, "% u ", loop_count );

        fprintf(io_ptr, "% u ", num_of_freqs );
        fprintf(io_ptr, "% u ", num_of_points );

        for (i=0; i < num_of_freqs; i++ )
                fprintf(io_ptr, "% .3f ", freq[i] );

        for (i=0; i < num_of_points; i++ )
                fprintf(io_ptr, "% .1f ", point[i] );

        for (i=0; i<10; i++) fprintf(io_ptr,"%c",26);        /* eof's */


        fclose(io_ptr);
        clear_status();
}

/***********************************************************************
*

        Save aquired data to a file.

************************************************************************
*/

aquired_to_file()

{
```

```
long count;
int i;
char b,c;
short *p;
short a;

        /* Print messages */

        clear_status();

        cprintf(stat_row+2, stat_col, purple, lgreen, 0, " Transfering from
");
        cprintf(stat_row+3, stat_col, purple, lgreen, 0, "  board to file ...
");


        if ((io_ptr = fopen(out_file,"wb")) == 0 )
         {
                cprintf(inf_row+2, inf_col, yellow, lred, 0, " Can't create o
utput file: ");
                cprintf(inf_row+3, inf_col, yellow, lred, 0, " < %s  >  ",out
_file);

                return;
         }

        setbuf (io_ptr, bigbuf);


        /* header */

        count = record_time * 1000;
        fprintf(io_ptr," \"This file contains %lu points.\"\n", count );


        /* out data points into file until all are in for selected range */

        p = ad_ram;

        for (i=0; i < count; i++)
         {
                a = *(p+i);

                b = a >> 8;
                c = a;

                putc(b ,io_ptr);
                putc(c ,io_ptr);
         }


        for (i=0; i<10; i++) fprintf(io_ptr,"%c",26);       /* eof's */
        fclose(io_ptr);

        clear_status();

}
```

```
/*****************************************************************************
*

        Load a specified amount of data to an array to graph

*****************************************************************************
*/

board_to_array ( a_ptr )

float a_ptr[];
{

long count,i;
unsigned short *p;
unsigned int temp;


        count = record_time * 1000;
        if (count > 50000) count = 50000;


        /* put data points into array until all are in for selected range */

        p = ad_ram;

        for ( i = 0; i < count; i++)
         {
                a_ptr[i] = ( *(p+i) & 0xfffe) >> 1;

                while (temp != a_ptr[i])
                 {
                        temp = a_ptr[i];
                        a_ptr[i] = ( *(p+i) & 0xfffe) >> 1;
                 }
         }

}



/***************************/
/*** go_av.c  boundpoint ***/
/***************************/


/*****************************************************************************

        Get the burst length

*****************************************************************************
/
```

```
get_burst_length()

{
float temp;

        cprintf(que_row+2, que_col, yellow, black, 0, " Enter burst time in m
msecs. ");
        cprintf(que_row+3, que_col, yellow, black, 0, "    Range is 0.0 - 409
80.0 : ");

        scanf("%f", &temp);
        total_time = temp;

        clear_questions();

        cprintf(sf2_row+3, sf2_col, white, blue, 0, "<                    >");
        cprintf(sf2_row+3, sf2_col+2, white, blue, 0, "%.1f msecs", temp);
}

/************************************************************************

*       Get the ramp length

*************************************************************************
/

get_ramp()

{
float temp;

        cprintf(que_row+2, que_col, yellow, black, 0, " Enter ramp time in ms
ecs. ");
        cprintf(que_row+3, que_col, yellow, black, 0, "    Range is 0.001 - [illegible]
0.0   : ");

        scanf("%f", &temp);
        ramp_time = temp;

        clear_questions();

        cprintf(sf2_row+5, sf2_col, white, blue, 0, "<                    >");
        cprintf(sf2_row+5, sf2_col+2, white, blue, 0, "%.1f msecs", temp);
}

/************************************************************************

        Get the decay length

*************************************************************************
/

get_decay()

{
float temp;
```

```
        cprintf(que_row+2, que_col, yellow, black, 0, " Enter decay time in m
secs. ");
        cprintf(que_row+3, que_col, yellow, black, 0, "    Range is 0.001 - 1
0.0    : ");

        scanf("%f", &temp);
        decay_time = temp;

        clear_questions();

        cprintf(sf2_row+7, sf2_col, white, blue, 0, "<                   >");
        cprintf(sf2_row+7, sf2_col+2, white, blue, 0, "%.1f msecs", temp);
}

/*****************************************************************************

        Get the mix lag time

*****************************************************************************
/

get_mix_lag()

{
float temp;

        cprintf(que_row+2, que_col, yellow, black, 0, " Enter mix lag time in
msecs ");
        cprintf(que_row+3, que_col, yellow, black, 0, "   before the 20 Hz p
oint:  ");

        scanf("%f", &temp);
        mix_lag = temp;

        clear_questions();

        cprintf(sf2_row+9, sf2_col, white, blue, 0, "<                   >");
        cprintf(sf2_row+9, sf2_col+2, white, blue, 0, "%.1f msecs", temp);
}


/*****************************************************************************

        Get the amount of time to record for

*****************************************************************************
/

get_record_time()

{
float temp;
```

```
        cprintf(que_row+2, que_col, yellow, black, 0, " Enter record time in
msecs. ");
        cprintf(que_row+3, que_col, yellow, black, 0, "    Range is 0.0 - 131
.0    : ");

        scanf("%f", &temp);
        record_time = temp;

        clear_questions();

        cprintf(sf2_row+1, sf2_col, white, blue, 0, "<                  >");
        cprintf(sf2_row+1, sf2_col+2, white, blue, 0, "%.1f msecs", temp);
}


/***********************************************************************

        Get the amount of time to delay before recording

************************************************************************
/

get_delay_time()

{
float temp;

        cprintf(que_row+2, que_col, yellow, black, 0, " Enter delay time in m
secs. ");
        cprintf(que_row+3, que_col, yellow, black, 0, "    Range is 0.0 - 131
.0    : ");

        scanf("%f", &temp);
        delay_time = temp;

        clear_questions();

        cprintf(sf2_row+3, sf2_col, white, blue, 0, "<                  >");
        cprintf(sf2_row+3, sf2_col+2, white, blue, 0, "%.1f msecs", temp);
}


/**********************************************************************

        Get the data to write out of the DAC

**********************************************************************/

get dac_out_data ()
{
        cprintf(que_row+2, que_col, yellow, black, 0, "Enter hex value (0 - F
FFF)");
        cprintf(que_row+3, que_col, yellow, black, 0, "to put out of the DAC:
");
        scanf("%x", &test_dac_out);
```

```
        cprintf(sf2_row+1, sf2_col, white, blue, 0, "<                    >");
        cprintf(sf2_row+1, sf2_col+3, white, blue, 0, "%x",test_dac_out);

        clear_questions ();
}

/***************************************************************************
*

          The action vector (AV) section must have certain routines included
        before it in order that pointers to the routines called will be
        properly initialized.

****************************************************************************
*/


/***************************************************************************

        Stuff to put before go_av for use in av structure without having to
include whole operations there. (convienience factor)

****************************************************************************
/

/***************************************************************************

        Based on active position, do the proper file transfer.

****************************************************************************
/

file_transfer_go()

{

        if (current_sm.row[0] == 1)          /* parms to file */
         {
                parms_to_file();
         }
        else if (current_sm.row[1] == 1)    /* file to parms */
         {
                file_to_parms();
         }
        else if (current_sm.row[2] == 1)    /* aquired to file */
         {
                aquired_to_file();
         }

        zero_sel_matrix();
}



/***************************************************************************
```

```
        calculate values for looping. and generate the wave.

*****************************************************************************
/

calc_wave()

{
        if (xmit_mode == 1)
         {
                calc_sweep_vals();
                wave_gen2();
         }
        else
         {
                calc_loop_vals();
                wave_gen();
         }
}




/****************************************************************************

        Adjust the mix lag up or down from within board control menu

*****************************************************************************
/

adjust_mix_lag()

{
int choice;
int test_go;

        clear_messages();
        pr_adj_ms();

        test_go = 1;
        while ( test_go )          /* do until return_key hit (once a key hit)
*/
        {

                /* print value */

                cprintf(sf2_row+6, sf2_col, white, blue, 0, "<
>");
                cprintf(sf2_row+6, sf2_col+2, white, blue, 0, "%.1f msecs", m
ix_lag);


                /* get new key (should we get more or quit) */

                choice = akeynum( getkey() );
                switch (choice)
                {
```

```
                        case 5:  test_go = 0; break;  /* return key */
                        case 6:  mix_lag += 0.1; break;  /* plus key */
                        case 7:  mix_lag -= 0.1; break;  /* minus key */
                        default:  break;
                }
        }

        pr_messages();
}


/*****************************************************************************

        Calculate the mix lag constants

******************************************************************************
/

cmlcs()

{
float t_5khz, delay;

        /* find and set freq time lag and delay for loading addr 1 & 2 */

        t_5khz = 5000 / freq_per_time ;  /* points */
        delay = t_5khz - (mix_lag * 1000); /* points (or micro secs) */

        mix_lag_cnt2 = 0x0ffffff;       /* default */

        if (delay < 0.0)
         {
                mix_lag_cnt2 += delay;                  /* trans before mix */
                delay = 0.0;                            /* trans starts at 0, or cl
ose anyway */
         }

        mix_lag_cnt = 0x0ffffff - delay;        /* val put in addr1 */

}


/*****************************************************************************

        Load transmit control parms and go

******************************************************************************
/

transmit()

{
```

```
short *p;

        load_loop_count();

        /* current version has no difference between sweep and discrete here
*/

        load_addr1( 0.0 );
        load_loop_start();

        p = ad_out1;
        *p = go_val;

        play();
}

/*************************************************************************

        Initiate a transmit and a recieve

**************************************************************************
/

trans_rec()
{

        load_delay_time();
        load_record_time();

        transmit();
        recieve();
}


/*************************************************************************

        Initiate a recieve

**************************************************************************
/

recieve()
{
short *p;

        p = ad_start;
        *p = go_val;
}

/*************************************************************************

        Load parms and initiate a reciept without a prior transmit

**************************************************************************
/
```

```
recieve_only()
{
short *p;

        load_delay_time();
        load_record_time();

        p = ad_start;
        *p = go_val;
}


/*************************************************************************

        Move the recorded data to the area for retransmission

**************************************************************************
/

ad_to_da()
{
short *p1,*p2;
int i,a;

        p1 = ad_ram;
        p2 = da_ram;


        /* move the number of points in recorded time from AD ram to DA ram *
/

        a =  record_time * 1000;
        for (i=0; i< a; i++) *p2++ = (*p1++ << 3) & 0xfff0;

        *p2 = 0x8001;  /* stop bit set */
}

/*************************************************************************

        indirect call to get frequencies

**************************************************************************
/
get_freqs_ind()
{
        get_freqs();
}


/***********************************************************************

        Print appropriate menu based on position of keystroke

***********************************************************************/

menu_select_go ()
{
```

```
int i;

        /* Print the menu corresponding to the row selected */

        for ( i = 0; i < max_row; i++ )
        {
                if (current_sm.row[i] == 1)
                {
                        pr_menu (i+1);
                        zero_sel_matrix ();
                        return;
                }
        }
}

/*****************************************************************************

    Clear the selection matrix

*****************************************************************************/

zero_sel_matrix ()
{
int i;

        current_sm.menu = 0;

        for (i=0; i < max_row; i++)
                current_sm.row[i] = 0;

        for (i=0; i < max_col; i++)
                current_sm.col[i] = 0;
}


/*****************************************************************************

        UP_MENU- moves the position to the first place in the menu of a
level one above the current one, and then prints the new menu.

*****************************************************************************/

up_menu ()
{
        zero_sel_matrix();

        if ((pos.menu < max_menu) && (pos.menu > 0))
        {
                pos.menu  = 0;
                pos.row   = 0;
                pos.col   = 0;

                pr_menu (pos.menu);
        }

        else if (pos.menu == 0) exit_sys();
```

```
.}


exit_sys()
{
        clear_screen();
        vmode(3);
        exit(0);
}

/*************************************************************************

        Sets the logical position to reflect the desired choice

*************************************************************************/


go_to_quit ()
{
        pos.col = quit_col;
        pos.row = quit_row;
}


go_to_go ()
{
        pos.col = go_col;
        pos.row = go_row;
}


go_to_first ()
{
        pos.col = first_col;
        pos.row = first_row;
}


/*************************************************************************

        move one space in the proper direction

*************************************************************************/

inc_row ()
{
        pos.row += 1;
}

dec_row ()
{
        pos.row -= 1;
```

```
}

/***********************************************************************

        HILIGHT and UNHILIGHT -

                These routines perform the switch function in order to
        hilight and unhilight a field in a menu selection situation.

***********************************************************************/

hilight ()
{
        change_pos_color ( hl_color_bg, hl_color_fg );
}


unhilight ()
{
        change_pos_color ( uhl_color_bg, uhl_color_fg );
}

/***********************************************************************

        Mark place in the selection matrix as chosen.

***********************************************************************/

activate_pos ()
{
        /* mark as selected: current menu, row, and column */

        current_sm.menu = pos.menu;
        current_sm.row [pos.row] = 1;
        current_sm.col [pos.col] = 1;

}



/***********************************************************************

        Get input filename

***********************************************************************/

get_filename_in ()
{
        /*  Print question */

        cprintf(que_row+2, que_col, yellow, black, 0, "Enter filename: ");
        scanf("%s",in_file);

        clear_questions ();
```

```
	cprintf(sf2_row+3, sf2_col, white, blue, 0, "<                          >");
	cprintf(sf2_row+3, sf2_col+2, white, blue, 0, "%s", in_file);
}

/************************************************************************

	Get output filename

************************************************************************/

get_filename_out ()

{
int place;

	/*  Print question */

	cprintf(que_row+2, que_col, yellow, black, 0, "Enter filename: ");
	scanf("%s",out_file);

	clear_questions ();


	/* Print filename in proper place */

	if (current_sm.row[0] == 1)          place = sf2_row+1;
	else if (current_sm.row[2] == 1) place = sf2_row+5;

	cprintf(place, sf2_col, white, blue, 0, "<                          >");
	cprintf(place, sf2_col+2, white, blue, 0, "%s", out_file);
}

/************************************************************************

	Plot the data desired

************************************************************************/

data_to_screen()

{
	plot_data();

	clear_screen();
	vmode(16);

	pr_common();
	pr_menu4();
}



/************************************************************************

	 Test the A to D converter by pulling in current values until
```

done.

```
*************************************************************************/

test_adc_go ()
{
int choice;
int test_go;
short *p;
short temp;

        /* Print messages */

        clear_status();
        cprintf(stat_row+2, stat_col, purple, lgreen, 0, " Testing A to D ...
");

        clear_messages();
        pr_test_ms();

        p = ad_read;
        test_go = 1;
        while ( test_go )           /* do until end_key hit (once a key hit) */
        {

                /* get an A to D value and print it (bits 1-12)*/

                temp = *p >> 1;

                cprintf(inf_row+3, inf_col, lgreen, black, 0,
                        " INPUT value is: %8x",temp & 0x0fff);


                /* get new key (should we get more or quit) */

                choice = keynum( getkey() );
                switch (choice)
                {
                        case 4:  test_go = 0; break;  /* end key */

                        default:  break;
                }
        }

        pr_messages();
        clear_status();
        clear_info();
}

/************************************************************************

          Test the D to A converter by putting out specified values
        until done.

*************************************************************************/
```

```
test_dac_go ()
{
int choice;
int test_go;
short *p;

        /* Print messages */

        clear_status();
        cprintf(stat_row+2, stat_col, purple, lgreen, 0, " Testing D to A ...
");

        clear_messages();
        pr_test_dac_ms();

        p = da_out;
        test_go = 1;
        while ( test_go )          /* do until end or return hit (once a key h
it) */
        {

                *p = test_dac_out;
                                            /
                cprintf(inf_row+2, inf_col, lgreen, black, 0, " Value put out
is:       ");
                cprintf(inf_row+2, inf_col, lgreen, black, 0, " Value put out
is: %x",test_dac_out);

                choice = akeynum( getkey() );

                switch (choice)
                {
                        case 4:  test_go = 0; break;            /* end key */

                        case 5:  get_dac_out_data(); break; /* return key */

                        case 6:  test_dac_out += 0x10; break;   /* plus key   *
/

                        case 7:  test_dac_out -= 0x10; break;   /* minus key
*/

                        default:  break;
                }
        }

        pr_messages();
        clear_status();
        clear_info();
}



/*****************************************************************************
```

```
        Action Vector initialization

          Basically, this section is a large structure of pointers to routine
s.
        They are organized so as to allow the chaining together of routines
        into action chains for meaningful menu driven operations control.

          There is also information contained as to what position
        in the menu the action is pertaining to, and what the physical layout
        of the menu is at that point.

*****************************************************************************
/

struct position info  location [max_menu] [max_row] [max_col] =

{

/*
        Bracket & data labels

    |  |  |  | |   |  |  |  |  |
    |  |  |  | | A | A| A| |   |
    |  |  |  |L| b | b| c| |   |        (See structures of:
    |  |  |  |e| s | s|  |C|   |                 position,
    | M|  |  |n|   |  | V|o| F|                  action_vector,
    | e| R| C|g| R | C| e|u| u|                  and position_info
    | n| o| o|t| o | o| c|n| n|             for further edification.)
    | u| w| l|h| w | l| t|t| c|
      *  *  * *  *   *  * *  *                                            */
      {
         {  {
                0                   /* null vector */
                     },

             {
               f2_leng, sf1_row, box_col+13,

/* down   */            { { 3, { unhilight, inc_row, hilight     } },
/* up     */              { 0 },
/* right  */              { 0 },
/* left   */              { 3, { unhilight, go_to_quit, hilight  } },
/* end    */              { 3, { unhilight, go_to_quit, hilight  } },
/* return */              { 2, { activate_pos, menu_select_go    } } } } },


         {  {
               f1_leng, sf1_row, box_col+1,

/* down   */            { { 0 },
/* up     */              { 0 },
/* right  */              { 3, { unhilight, go_to_first, hilight } },
/* left   */              { 0 },
/* end    */              { 0 },
/* return */              { 1, { exit_sys                          } } } },
```

```
            {
              f2_leng, sf2_row+1, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight      } },
/* up     */            { 3, { unhilight, dec_row, hilight      } },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_quit, hilight  } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 2, { activate_pos, menu_select_go    } } } } },

        {  {
             0                  /* null vector */
                 },

            {
              f2_leng, sf2_row+2, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight      } },
/* up     */            { 3, { unhilight, dec_row, hilight      } },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_quit, hilight  } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 2, { activate_pos, menu_select_go    } } } } },

        {  {
             0                  /* null vector */
                 },

            {
              f2_leng, sf2_row+3, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight      } },
/* up     */            { 3, { unhilight, dec_row, hilight      } },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_quit, hilight  } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 2, { activate_pos, menu_select_go    } } } } },


        {  {
             0                  /* null vector */
                 },

            {
              f2_leng, sf2_row+4, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight      } },
/* up     */            { 3, { unhilight, dec_row, hilight      } },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_quit, hilight  } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 2, { activate_pos, menu_select_go    } } } } },
```

```
        {  {
               0                /* null vector */
                   },

              {
                f2_leng, sf2_row+5, box_col+13,

/* down   */          { { 0 },
/* up     */            { 3, { unhilight, dec_row, hilight     } },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_quit, hilight  } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 2, { activate_pos, menu_select_go    } } } } } },



              /********** Menu #1 - (Xmit Parms) ***********/

      {  {  {
                f1_leng, sf1_row, box_col+1,

/* down   */          { { 3, { unhilight, inc_row, hilight     } },
/* up     */            { 0 },
/* right  */            { 3, { unhilight, go_to_first, hilight } },
/* left   */            { 0 },
/* end    */            { 3, { unhilight, go_to_quit , hilight } },
/* return */            { 1, { calc_wave                       } } } },

              {
                f2_leng, sf2_row, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight     } },
/* up     */            { 0 },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_go, hilight    } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 1, { get_freqs_ind                   } } } } },

          {  {
                f1_leng, sf1_row+1, box_col+1,

/* down   */          { { 0 },
/* up     */            { 3, { unhilight, dec_row, hilight     } },
/* right  */            { 3, { unhilight, go_to_first, hilight } },
/* left   */            { 0 },
/* end    */            { 0 },
/* return */            { 1, { up_menu                         } } } },

              {
                f2_leng, sf2_row+2, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight     } },
/* up     */            { 3, { unhilight, dec_row, hilight     } },
```

```
/* right  */              { 0 },
/* left   */              { 3, { unhilight, go_to_go, hilight    } },
/* end    */              { 3, { unhilight, go_to_quit, hilight  } },
/* return */              { 1, { get_burst_length                } } } } },


        {  {
              0                  /* null vector */
                 },

           {
             f2_leng, sf2_row+4, box_col+13,

/* down   */            { { 3, { unhilight, inc_row, hilight     } },
/* up     */              { 3, { unhilight, dec_row, hilight     } },
/* right  */              { 0 },
/* left   */              { 3, { unhilight, go_to_go, hilight    } },
/* end    */              { 3, { unhilight, go_to_quit, hilight  } },
/* return */              { 1, { get_ramp                        } } } } },


        {  {
              0                  /* null vector */
                 },

           {
             f2_leng, sf2_row+6, box_col+13,

/* down   */            { { 3, { unhilight, inc_row, hilight     } },
/* up     */              { 3, { unhilight, dec_row, hilight     } },
/* right  */              { 0 },
/* left   */              { 3, { unhilight, go_to_go, hilight    } },
/* end    */              { 3, { unhilight, go_to_quit, hilight  } },
/* return */              { 1, { get_decay                       } } } } },


        {  {
              0                  /* null vector */
                 },

           {
             f2_leng, sf2_row+8, box_col+13,

/* down   */            { { 0 },
/* up     */              { 3, { unhilight, dec_row, hilight     } },
/* right  */              { 0 },
/* left   */              { 3, { unhilight, go_to_go, hilight    } },
/* end    */              { 3, { unhilight, go_to_quit, hilight  } },
/* return */              { 1, { get_mix_lag                     } } } } },


        {  {
              0                  /* null vector */
                 },
```

```
            {
                0                   /* null vector */
                    }   }   },


                /********** Menu #2 - (Reception Parms) ************/

        {
            {   {
                0                   /* null vector */
                    },


            {
              f2_leng, sf1_row, box_col+13,

/* down    */            { { 3, { unhilight, inc_row, hilight      } },
/* up      */              { 0 },
/* right   */              { 0 },
/* left    */              { 3, { unhilight, go_to_quit, hilight  } },
/* end     */              { 3, { unhilight, go_to_quit, hilight  } },
/* return */               { 1, { get_record_time                  } } } } },


            {   {
              f1_leng, sf1_row, box_col+1,

/* down    */            { { 0 },
/* up      */              { 0 },
/* right   */              { 3, { unhilight, go_to_first, hilight } },
/* left    */              { 0 },
/* end     */              { 0 },
/* return */               { 1, { up_menu                        } } } },


            {
              f2_leng, sf1_row+2, box_col+13,

/* down    */           { { 0 },
/* up      */              { 3, { unhilight, dec_row, hilight      } },
/* right   */              { 0 },
/* left    */              { 3, { unhilight, go_to_quit, hilight  } },
/* end     */              { 3, { unhilight, go_to_quit, hilight  } },
/* return */               { 1, { get_delay_time                   } } } } },



            {   {
                0                   /* null vector */
                    },

            {
                0                   /* null vector */
                    }   },
```

```
        {  {
               0                    /* null vector */
                 },

           {
               0                    /* null vector */
                 }  },


        {  {
               0                    /* null vector */
                 },

           {
               0                    /* null vector */
                 }  },


        {  {
               0                    /* null vector */
                 },

           {
               0                    /* null vector */
                 }  }  },



               /********** Menu #3 - (Board Control) ************/
    {
        {  {
               0                    /* null vector */
                 },

           {
             f2_leng, sf1_row, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight     } },
/* up     */            { 0 },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_quit, hilight  } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 1, { transmit                         } } } } },

        {  {
             f1_leng, sf1_row, box_col+1,

/* down   */          { { 0 },
/* up     */            { 0 },
```

```
/* right  */            { 3, { unhiliant, go to first, nilight } },
/* left   */            { 0 },
/* end    */            { 0 },
/* return */            { 1, { up menu                      } } } },

        {
          f2_leng, sf1_row+1, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight     } },
/* up     */            { 3, { unhilight, dec_row, hilight     } },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_quit, hilight  } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 1, { recieve_only                     } } } } },

      {   {
            0                  /* null vector */
                },

        {
          f2_leng, sf1_row+2, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight     } },
/* up     */            { 3, { unhilight, dec_row, hilight     } },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_quit, hilight  } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 1, { trans_rec                 } } } } },


      {   {
            0                  /* null vector */
                },

        {
          f2_leng, sf1_row+3, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight     } },
/* up     */            { 3, { unhilight, dec_row, hilight     } },
/* right  */            { 0 },
/* left   */            { 3, { unhilight, go_to_quit, hilight  } },
/* end    */            { 3, { unhilight, go_to_quit, hilight  } },
/* return */            { 5, { zero_ad, load_record_time,
                               unhilight, dec_row, hilight    } } } } },


      {   {
            0                  /* null vector */
                },


        {
          f2_leng, sf1_row+4, box_col+13,

/* down   */          { { 3, { unhilight, inc_row, hilight     } },
```

```
/* up      */            { 3, { unhilight, dec_row, hilight      } },
/* right   */            { 0 },
/* left    */            { 3, { unhilight, go_to_quit, hilight   } },
/* end     */            { 3, { unhilight, go_to_quit, hilight   } },
/* return  */            { 5, { ad_to_da, unhilight, dec_row,
                                   dec_row, hilight       } } } } },


        {  {
              0                   /* null vector */
                  },

           {
             f2_leng, sf1_row+5, box_col+13,

/* down    */           { { 0 },
/* up      */             { 3, { unhilight, dec_row, hilight      } },
/* right   */             { 0 },
/* left    */             { 3, { unhilight, go_to_quit, hilight   } },
/* end     */             { 3, { unhilight, go_to_quit, hilight   } },
/* return  */             { 1, { adjust_mix_lag                     } } } } },



              /*********** Menu #4   (Data Transfer) ***********/

     {  {  {
             f1_leng, sf1_row, box_col+1,

/* down    */           { { 3, { unhilight, inc_row, hilight      } },
/* up      */             { 0 },
/* right   */             { 3, { unhilight, go_to_first, hilight } },
/* left    */             { 0 },
/* end     */             { 3, { unhilight, go_to_quit , hilight } },
/* return  */             { 1, { file_transfer_go               } } } },

           {
             f2_leng, sf2_row, box_col+13,

/* down    */           { { 3, { unhilight, inc_row, hilight      } },
/* up      */             { 0 },
/* right   */             { 0 },
/* left    */             { 3, { unhilight, go_to_go, hilight     } },
/* end     */             { 3, { unhilight, go_to_quit, hilight   } },
/* return  */             { 2, { activate_pos, get_filename_out   } } } } },

        {  {
             f1_leng, sf1_row+1, box_col+1,

/* down    */           { { 0 },
/* up      */             { 3, { unhilight, dec_row, hilight      } },
```

```
/* right  */              { 3, { unhilight, go_to_first, hilight } },
/* left   */              { 0 },
/* end    */              { 0 },
/* return */              { 1, { up_menu                        } } } },

          {
            f2_leng, sf2_row+2, box_col+13,

/* down   */            { { 3, { unhilight, inc_row, hilight      } },
/* up     */              { 3, { unhilight, dec_row, hilight      } },
/* right  */              { 0 },
/* left   */              { 3, { unhilight, go_to_go, hilight     } },
/* end    */              { 3, { unhilight, go_to_quit, hilight   } },
/* return */              { 2, { activate_pos, get_filename_in    } } } } },


        {  {
             0                    /* null vector */
                 },

          {
            f2_leng, sf2_row+4, box_col+13,

/* down   */            { { 3, { unhilight, inc_row, hilight      } },
/* up     */              { 3, { unhilight, dec_row, hilight      } },
/* right  */              { 0 },
/* left   */              { 3, { unhilight, go_to_go, hilight     } },
/* end    */              { 3, { unhilight, go_to_quit, hilight   } },
/* return */              { 2, { activate_pos, get_filename_out   } } } } },


        {  {
             0                    /* null vector */
                 },

          {
            f2_leng, sf2_row+7, box_col+13,

/* down   */           { { 0 },
/* up     */              { 3, { unhilight, dec_row, hilight      } },
/* right  */              { 0 },
/* left   */              { 3, { unhilight, go_to_go, hilight     } },
/* end    */              { 3, { unhilight, go_to_quit, hilight   } },
/* return */              { 1, { data_to_screen                   } } } } },


        {  {
             0                    /* null vector */
                 },

          {
             0                    /* null vector */
                 }  },
```

```
        { {
              0                 /* null vector */
                  },

          {
              0                 /* null vector */
                  }  }  },


            /********** Menu #5 - (Test DAC) ***********/

     {  {  {
              f1_leng, sf1_row, box_col+1,

/* down    */          { { 3, { unhilight, inc_row, hilight     } },
/* up      */            { 0 },
/* right   */            { 3, { unhilight, go_to_first, hilight } },
/* left    */            { 0 },
/* end     */            { 3, { unhilight, go_to_quit, hilight  } },
/* return  */            { 1, { test_dac_go                     } } } },


          {
              f2_leng, sf1_row, box_col+13,

/* down    */          { { 0 },
/* up      */            { 0 },
/* right   */            { 0 },
/* left    */            { 3, { unhilight, go_to_go, hilight    } },
/* end     */            { 3, { unhilight, go_to_quit, hilight  } },
/* return  */            { 5, { activate_pos, get_dac_out_data,
                                unhilight, go_to_go, hilight    } } } } },


        {  {
              f1_leng, sf1_row+1, box_col+1,

/* down    */          { { 0 },
/* up      */            { 3, { unhilight, dec_row, hilight     } },
/* right   */            { 3, { unhilight, go_to_first, hilight } },
/* left    */            { 0 },
/* end     */            { 0 },
/* return  */            { 1, { up_menu                         } } } },

          {
              0                 /* null vector */
                  }  },


        {  {
```

```
                O                   /* null vector */
                    },
            {
                O                   /* null vector */
                    }   },


        {   {
                O                   /* null vector */
                    },
            {
                O                   /* null vector */
                    }   },


        {   {
                O                   /* null vector */
                    },
            {
                O                   /* null vector */
                    }   },


        {   {
                O                   /* null vector */
                    },
            {
                O                   /* null vector */
                    }   }   },



                /********** Menu #6 - (Test ADC) ************/

    {
        {   {
                O                   /* null vector */
                    },

            {
              f2_leng, sf1_row, box_col+13,
/* down    */          { { O },
/* up      */            { O },
/* right   */            { O },
/* left    */            { 3, { unhilight, go_to_quit, hilight  } },
/* end     */            { 3, { unhilight, go_to_quit, hilight  } },
```

```
/* return */              { 1, { test_adc_go                     } } } } },

          {  {
                fl_leng, sfl_row, box_col+1,

/* down    */           { { 0 },
/* up      */             { 0 },
/* right   */             { 3, { unhilight, go_to_first, hilight } },
/* left    */             { 0 },
/* end     */             { 0 },
/* return  */             { 1, { up_menu                         } } } },

             {
                0                   /* null vector */
                    }  },

          {  {
                0                   /* null vector */
                    },

             {
                0                   /* null vector */
                    }  },

          {  {
                0                   /* null vector */
                    },

             {
                0                   /* null vector */
                    }  },

          {  {
                0                   /* null vector */
                    },

             {
                0                   /* null vector */
                    }  },

          {  {
                0                   /* null vector */
                    },

             {
                0                   /* null vector */
```

```
                        }  }  }

};



/***************************************************************************

        Take Action-

                This routine takes in a given key and the position for
        which that key was hit, and causes the proper chain of actions
        to be carried out.

***************************************************************************/

take_action ( key )

int key;
{
int count,index;
struct position_info  *locus;

        locus = &location [pos.menu] [pos.row] [pos.col];

        count = locus->act_vect[key].count;
        index = 0;

        while ( count-- > 0 )
        {
                (*locus->act_vect[key].func[index++]) ();
        }

}


/***************************************************************************

        DOS call to get the keyboard key value

***************************************************************************/

int getkey ()
{
        inregs.ax = 0x0000;          /* use read keyboard character   */
        INT86(0x16,&inregs );        /* service                        */

        return (inregs.ax);          /* return the 16 bit key   */
}


/***************************************************************************
```

```
        Change the color of the space at the current position.

****************************************************************************/

change_pos_color ( color_bg, color_fg )

int color_bg, color_fg;
{
struct position_info  *locus;
int  times, row, col;

        locus = &location [pos.menu] [pos.row] [pos.col];

        times = locus->length;
        row = locus->abs_row;
        col = locus->abs_col;

        while (times-- > 0)                         /* for the length of field
*/
        {
                curset ( row, col++, page);

                inregs.ax = 0x0800;                 /* read the character and
*/
                inregs.bx = page << 8;              /* and it's attribute
*/
                INT86(0x10,&inregs );


                inregs.ax = inregs.ax & 0x00ff;  /* write back same char     */
                inregs.ax |= 0x0900;
                inregs.bx = inregs.bx & 0xff00;  /* same page */
                inregs.bx |= ((color_bg << 4)
                            + color_fg);         /* different attributes    */
                inregs.cx = 1;                    /* one copy (no repeats)    *
/
                INT86(0x10,&inregs );
        }
}

/****************************************************************************

        KEYNUM- returns an integer value to be used as an index for each
meaningful key.

****************************************************************************/

int keynum (key)

int key;
{
int temp;

        switch (key)
        {
                case down_key:
                                        temp=0; break;
```

```
                case up_key:
                                temp=1; break;

                case right_key:
                                temp=2; break;

                case left_key:
                                temp=3; break;

                case end_key:
                                temp=4; break;

                case return_key:
                                temp=5; break;

                default:
                                temp=0;
        }
        return (temp);
}

/***************************************************************************

        AKEYNUM- returns an integer value to be used as an index for each
meaningful key. (A for Alternate)

***************************************************************************/

int akeynum (key)

int key;
{
int temp;

        switch (key)
        {
                case down_key:
                                temp=0; break;

                case up_key:
                                temp=1; break;

                case right_key:
                                temp=2; break;

                case left_key:
                                temp=3; break;

                case end_key:
                                temp=4; break;

                case return_key:
                                temp=5; break;

                case plus_key:
                                temp=6; break;
```

```
		case minus_key:
					temp=7; break;

		default:
					temp=0;
	}
	return (temp);
}

/****************************/
/***    pr.c  boundpoint  ***/
/****************************/

/***************************************************************************

	PR_MENU- prints the menu corresponding to the menu number passed in.

***************************************************************************/

pr_menu (menu_num)

int menu_num;
{
	switch (menu_num)
	{
		case 0:
				pr_menu0 (); break;

		case 1:
				pr_menu1 (); break;

		case 2:
				pr_menu2 (); break;

		case 3:
				pr_menu3 (); break;

		case 4:
				pr_menu4 (); break;

		case 5:
				pr_menu5 (); break;

		case 6:
				pr_menu6 (); break;

		default:
				break;
	}
}


/***************************************************************************

	Print those sections common to all of the main menus.
```

```
******************************************************************************/

pr_common ()
{
int i;

        /* Print in text mode, on page 0 */

        vpage(0);
        vmode(16);


        /* Print headers */

        cprintf(mess_row, mess_col, red, black, 0, "MESSAGES");
        cprintf(que_row, que_col, red, black, 0, "QUESTIONS");
        cprintf(act_row, act_col, red, black, 0, "ACTION");
        cprintf(sel_row, sel_col, red, black, 0, "SELECTION");
        cprintf(stat_row, stat_col, red, black, 0, "STATUS");

        /* Print messages */

        pr_messages();

        /* Print horizontal bars */

        cprchar(mess_row+1, mess_col, brown, black, 0, mess_und_leng, hzt_d);
        cprchar(que_row+1, que_col, brown, black, 0, que_und_leng, hzt_d);
        cprchar(stat_row+1, stat_col, brown, black, 0, stat_und_leng, hzt_d);

        cprchar(box_row,  box_col+1, m_bar_color_fg, m_bar_color_bg, 0, f1_le
ng, hzt_d);
        cprchar(box_row+12, box_col+1, m_bar_color_fg, m_bar_color_bg, 0, f1_
leng, hzt_d);
        cprchar(box_row, box_col+13, m_bar_color_fg, m_bar_color_bg, 0, f2_le
ng, hzt_d);
        cprchar(box_row+12, box_col+13, m_bar_color_fg, m_bar_color_bg, 0, f2
_leng, hzt_d);

        /* Print corners */

        cprchar(box_row, box_col, m_bar_color_fg, m_bar_color_bg, 0, 1, tl_d)
;
        cprchar(box_row, box_col+12, m_bar_color_fg, m_bar_color_bg, 0, 1, tm
_d);
        cprchar(box_row, box_col+38, m_bar_color_fg, m_bar_color_bg, 0, 1, tr
_d);
        cprchar(box_row+12, box_col, m_bar_color_fg, m_bar_color_bg, 0, 1, bl
_d);
        cprchar(box_row+12, box_col+12, m_bar_color_fg, m_bar_color_bg, 0, 1,
bm_d);
        cprchar(box_row+12, box_col+38, m_bar_color_fg, m_bar_color_bg, 0, 1,
br_d);

        /* Print vertical bars */

        for (i=box_row+1; i<box_row+12; i++)
```

```
        {       cprchar(i, box_col, m_bar_color_fg, m_bar_color_bg, 0, 1, vrt
_d);
                cprchar(i, box_col+12, m_bar_color_fg, m_bar_color_bg, 0, 1,
vrt_d);
                cprchar(i, box_col+38, m_bar_color_fg, m_bar_color_bg, 0, 1,
vrt_d);
        }
}

/*****************************************************************************

        Print standard messages concerning control keys

*****************************************************************************/

pr_messages ()
{
        cprintf(mess_row+2, mess_col, yellow, black, 0, "ARROWS -  Move in ar
rows direction");
        cprintf(mess_row+3, mess_col, yellow, black, 0, "RETURN -  Select opt
ion or action");
        cprintf(mess_row+4, mess_col, yellow, black, 0, "END    -  Go to quit
");
}


/*****************************************************************************

        Print menu 0 - The main menu

*****************************************************************************/

pr_menu0 ()
{

        /* Print color fields */
        blank_fields ();


        /* Print field choice labels */

        cprintf(sf1_row, sf1_col, opt_color_fg, opt_color_bg, 0, "EXIT");

        cprintf(sf1_row, sf2_col, opt_color_fg, opt_color_bg, 0, "TRANSMISSIO
N PARMS");
        cprintf(sf1_row+1, sf2_col, opt_color_fg, opt_color_bg, 0, "RECEPTION
PARMS");
        cprintf(sf1_row+2, sf2_col, opt_color_fg, opt_color_bg, 0, "BOARD CON
TROL");
        cprintf(sf1_row+3, sf2_col, opt_color_fg, opt_color_bg, 0, "DATA TRAN
SFER");
        cprintf(sf1_row+4, sf2_col, opt_color_fg, opt_color_bg, 0, "TEST DAC"
);
        cprintf(sf1_row+5, sf2_col, opt_color_fg, opt_color_bg, 0, "TEST ADC"
);
```

```
        /*  Print status */

        cprintf(stat_row+2, stat_col, yellow, black, 0, "At top menu. Selecti
ng");
        cprintf(stat_row+3, stat_col, yellow, black, 0, "function to carry ou
t.");

        /* Set current location */
        pos.menu = 0;
        pos.row = 0;
        pos.col = 1;

        /* and hilight current location */
        hilight ();

}

/***************************************************************************

        Print menu 1 - The frequency transmission selection menu

***************************************************************************/

pr_menu1 ()
{
        /* clear fields */
        blank_fields ();

        /* Print field choice labels */

        cprintf(sf1_row, sf1_col, opt_color_fg, opt_color_bg, 0, "CALC WAVE")
;
        cprintf(sf1_row+1, sf1_col, opt_color_fg, opt_color_bg, 0, "UP");

        cprintf(sf1_row, sf2_col, opt_color_fg, opt_color_bg, 0, "FREQUENCIES
");

        cprintf(sf1_row+2, sf2_col, opt_color_fg, opt_color_bg, 0, "BURST TIM
E");
        cprintf(sf1_row+3, sf2_col, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+3, sf2_col+2, opt_color_fg, opt_color_bg, 0, "%.1f ms
ecs", total_time);

        cprintf(sf1_row+4, sf2_col, opt_color_fg, opt_color_bg, 0, "RAMP UP")
;
        cprintf(sf1_row+5, sf2_col, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+5, sf2_col+2, opt_color_fg, opt_color_bg, 0, "%.1f ms
ecs", ramp_time);

        cprintf(sf1_row+6, sf2_col, opt_color_fg, opt_color_bg, 0, "DECAY");
        cprintf(sf1_row+7, sf2_col, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+7, sf2_col+2, opt_color_fg, opt_color_bg, 0, "%.1f ms
ecs", decay_time);
```

```
        cprintf(sf1_row+8, sf2_col, opt_color_fg, opt_color_bg, 0, "MIX LAG")
;
        cprintf(sf1_row+9, sf2_col, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+9, sf2_col+2, opt_color_fg, opt_color_bg, 0, "%.1f ms
ecs", mix_lag);

        /*  Print status */

        cprintf(stat_row+2, stat_col, yellow, black, 0, "Transmission paramet
er");
        cprintf(stat_row+3, stat_col, yellow, black, 0, "selection menu.  Sel
ect");
        cprintf(stat_row+4, stat_col, yellow, black, 0, "operating parameters
.");

        /* Set current location */
        pos.menu = 1;
        pos.row = 0;
        pos.col = 1;

        /* and hilight current location */
        hilight ();
}


/**********************************************************************

        Print menu 2 - The reception control menu

***********************************************************************/

pr_menu2 ()
{
        /* clear fields */
        blank_fields ();

        /* Print field choice labels */

        cprintf(sf1_row, sf1_col, opt_color_fg, opt_color_bg, 0, "UP");

        cprintf(sf1_row, sf2_col, opt_color_fg, opt_color_bg, 0, "RECORD TIME
");
        cprintf(sf1_row+1, sf2_col, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+1, sf2_col+2, opt_color_fg, opt_color_bg, 0, "%.1f ms
ecs", record_time);

        cprintf(sf1_row+2, sf2_col, opt_color_fg, opt_color_bg, 0, "DELAY TIM
E");
        cprintf(sf1_row+3, sf2_col, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+3, sf2_col+2, opt_color_fg, opt_color_bg, 0, "%.1f ms
ecs", delay_time);
```

```
        /*  Print status */

        cprintf(stat_row+2, stat_col, yellow, black, 0, "Reception parameter"
);
        cprintf(stat_row+3, stat_col, yellow, black, 0, "selection menu.  Sel
ect");
        cprintf(stat_row+4, stat_col, yellow, black, 0, "operating parameters
.");

        /* Set current location */
        pos.menu = 2;
        pos.row = 0;
        pos.col = 1;

        /* and hilight current location */
        hilight ();
}


/**************************************************************************

        Print menu 3 - The board control menu

**************************************************************************/

pr_menu3 ()
{
        /* clear fields */

        blank_fields ();


        /* Print field choice labels */

        cprintf(sf1_row, sf1_col, opt_color_fg, opt_color_bg, 0, "UP");

        cprintf(sf1_row, sf2_col, opt_color_fg, opt_color_bg, 0, "TRANSMIT");
        cprintf(sf1_row+1, sf2_col, opt_color_fg, opt_color_bg, 0, "RECEIVE")
;
        cprintf(sf1_row+2, sf2_col, opt_color_fg, opt_color_bg, 0, "TRANS/REC
EIVE");
        cprintf(sf1_row+3, sf2_col, opt_color_fg, opt_color_bg, 0, "CLEAR A/D
RAM");
        cprintf(sf1_row+4, sf2_col, opt_color_fg, opt_color_bg, 0, "AD RAM ->
DA RAM");
        cprintf(sf1_row+5, sf2_col, opt_color_fg, opt_color_bg, 0, "ADJUST MI
X LAG");
        cprintf(sf1_row+6, sf2_col, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+6, sf2_col+2, opt_color_fg, opt_color_bg, 0, "%.1f ms
ecs", mix_lag);


        /*  Print status */
```

```
        cprintf(stat_row+2, stat_col, yellow, black, 0, "Board control menu.
Selecting");
        cprintf(stat_row+3, stat_col, yellow, black, 0, "actions for board pe
rformance.");

        /* Set current location */
        pos.menu = 3;
        pos.row = 0;
        pos.col = 1;

        /* and hilight current location */
        hilight ();
}



/***********************************************************************

        Print menu 4 - The data transfer control menu

***********************************************************************/

pr_menu4 ()
{
        /* clear fields */

        blank_fields ();


        /* Print field choice labels */

        cprintf(sf1_row, sf1_col, opt_color_fg, opt_color_bg, 0, "GO");
        cprintf(sf1_row+1, sf1_col, opt_color_fg, opt_color_bg, 0, "UP");

        cprintf(sf1_row, sf2_col-1, opt_color_fg, opt_color_bg, 0, "PARMS TO
FILE");
        cprintf(sf1_row+1, sf2_col-1, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+2, sf2_col-1, opt_color_fg, opt_color_bg, 0, "FILE TO
PARMS");
        cprintf(sf1_row+3, sf2_col-1, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+4, sf2_col-1, opt_color_fg, opt_color_bg, 0, "RAM DAT
A TO FILE");
        cprintf(sf1_row+5, sf2_col-1, opt_color_fg, opt_color_bg, 0, "<
>");
        cprintf(sf1_row+7, sf2_col-1, opt_color_fg, opt_color_bg, 0, "RAM DAT
A TO SCREEN");


        /*  Print status */

        cprintf(stat_row+2, stat_col, yellow, black, 0, "File transfer menu.
Selecting");
        cprintf(stat_row+3, stat_col, yellow, black, 0, "desired file transfe
r action.");
```

```
        /* Set current location */

        pos.menu = 4;
        pos.row = 0;
        pos.col = 1;

        /* and hilight current location */

        hilight ();
}

/****************************************************************************

        Print menu 5 - The test DAC control menu

****************************************************************************/

int menu5 ()
{
        /* clear fields */
        blank_fields ();

        /* Print field choice labels */

        cprintf(sf1_row, sf1_col, opt_color_fg, opt_color_bg, 0, "GO");
        cprintf(sf1_row+1, sf1_col, opt_color_fg, opt_color_bg, 0, "UP");

        cprintf(sf1_row, sf2_col, opt_color_fg, opt_color_bg, 0, "PUT VALUE O
F DAC");
        cprintf(sf1_row+1, sf2_col, opt_color_fg, opt_color_bg, 0, "<
>");


        /*  Print status */

        cprintf(stat_row+2, stat_col, yellow, black, 0, "D to A test menu. ")
;


        /* Set current location */
        pos.menu = 5;
        pos.row = 0;
        pos.col = 1;

        /* and hilight current location */
        hilight ();
}

/****************************************************************************

        Print menu 6 - The test ADC control menu

****************************************************************************/
```

```
pr_menu6 ()
{
        /* clear fields */
        blank_fields ();

        /* Print field choice labels */

        cprintf(sf1_row, sf1_col, opt_color_fg, opt_color_bg, 0, "UP");

        cprintf(sf1_row, sf2_col, opt_color_fg, opt_color_bg, 0, "READ VALUE
AT ADC");


        /*  Print status */

        cprintf(stat_row+2, stat_col, yellow, black, 0, "A to D test menu. ")
;


        /* Set current location */
        pos.menu = 6;
        pos.row = 0;
        pos.col = 1;

        /* and hilight current location */
        hilight ();
}


/***************************************************************************

        Print messages to be displayed during testing of adc.

***************************************************************************/

pr_test_ms ()
{
        cprintf(mess_row+2, mess_col, yellow, red, 0, " Hit END to stop test.
");
        cprintf(mess_row+3, mess_col, yellow, red, 0, " Hit RETURN for more v
alues. ");
}

/***************************************************************************

        Print messages to be displayed during testing of dac.

***************************************************************************/

pr_test_dac_ms ()
{
        cprintf(mess_row+2, mess_col, yellow, red, 0, " Hit END to stop test.
");
        cprintf(mess_row+3, mess_col, yellow, red, 0, " Hit RETURN for more v
alues.   ");
        cprintf(mess_row+4, mess_col, yellow, red, 0, " Hit PLUS or MINUS to
adjust. ");
```

```
}

/*****************************************************************************

        Print messages to be displayed during adjustments

*****************************************************************************/

pr_adj_ms ()
{
        cprintf(mess_row+2, mess_col, yellow, red, 0, " Hit PLUS or MINUS to
adjust.  ");
        cprintf(mess_row+3, mess_col, yellow, red, 0, " RETURN when done adju
sting.  ");
}



/*****************************/
/*** pr_utils.c  boundpoint ***/
/*****************************/


/*****************************************************************************

        Color PRint CHARacter -

                This routine prints a character at a place, with attributes,
        and with a repeat factor.

*****************************************************************************/

cprchar ( row, col, color_fg, color_bg, page, repeat_factor, char_to_print)

short   color_fg, color_bg, char_to_print;
int     row, col, page, repeat_factor;
{

        /* place the cursor in desired position */

        curset (row,col,page);


        /* use interrupt 0x10 to print character */

        inregs.ax = (0x0900 | (0x00ff & char_to_print));
        inregs.bx = (page << 8) | ((0x0f & color_bg) << 4) | (color_fg & 0x0f
);
        inregs.cx = repeat_factor;
        INT86(0x10,&inregs );

}

/*****************************************************************************
```

```
            This function will print the string STRING with associateu
          arguments at the location row, col, page and in the color
          specified by color. This works just like printf()

*****************************************************************************/

.printf(row,col,colorf,colorb,page,string,a1,a2,a3,a4,a5,a6,a7,a8)

int row,col,colorf,colorb,page;
char *string;
unsigned a1,a2,a3,a4,a5,a6,a7,a8;
{
char buf[100];
int i;

     sprintf(buf,string,a1,a2,a3,a4,a5,a6,a7,a8);
     for(i=0;(buf[i]) && (i<130);i++){
        curset(row,col++,page);                          /* Position cursor c
orrectly     */
        inregs.ax = 0x0900;                              /* Call BIOS to put
character    */
        inregs.ax = inregs.ax + (0x0ff & buf[i]);        /* The character to
print        */
        inregs.bx = page << 8;                           /* Page to print on
*/
        inregs.bx = inregs.bx + (0x00f & colorf);        /* Color value
*/
        inregs.bx = inregs.bx + ((0x00f & colorb) << 4); /* Color value
*/
        inregs.cx = 1;                                   /* number of charact
ers to print */
        INT86(0x10,&inregs );                        /* Do the interrupt
*/
     }
     curset(row,col,page);
     return(0);
}


/*****************************************************************************

      This function will set the video mode to the value passed.

*****************************************************************************/

int vmode(mode)
int mode;
{

        inregs.ax = 0;                                /* Use BIOS call 0 to set
mode    */
        inregs.ax = 0xff & mode;                     /* Video mode number
*/
        INT86(0x10,&inregs );                           /* Do the interrupt
*/
```

```
        return(mode);
}


/*************************************************************************

            This function will set the DISPLAY page to the value passed.

*************************************************************************/

vpage(page)
int page;
{

        inregs.ax = 0x0500;                       /* Call BIOS to set page *
*/
        inregs.ax = inregs.ax + (0xff & page);   /* set page #
*/
        INT86(0x10,&inregs );
 }


/*************************************************************************

            This function will set the cursor to the position row, col,
            and page.

*************************************************************************/

cursset(row,col,page)
int row,col,page;
{

        inregs.ax = 0x0200;                      /* Call BIOS to set cursor   */
        inregs.dx = row << 8;                    /* set row number            */
        inregs.dx = inregs.dx + (0xff + col);   /* set col number
*/
        inregs.bx = 0xff & page;                 /* set page number           */
        INT86(0x10,&inregs );
 }


/*************************************************************************

            This function will return the current video mode, an integer
        from 1 to 16.

*************************************************************************/

int getmode()
{

        inregs.ax = 0x0F00;                   /* Use BIOS call 15 to get mode
*/
```

```
        INT86(0x10,&inregs );   /* CALL 0x10 is the video stuff  */
                                /* ah = # of columns              */
                                /* al = current display mode      */
                                /* bh = current display page      */
        return(inregs.ax);
}

/*************************************************************************

    This function will set the foreground color to the value passed.

*************************************************************************/

foregnd(color)
int color;
{

        inregs.ax = 0x0D00;                        /* Call BIOS to set fo
regnd  */
        inregs.bx = 0x0100;                       /* foregnd number = 1
*/
        inregs.bx = inregs.bx + (0xff &  color);      /* Color number
*/
        INT86(0x10,&inregs );
}

/*************************************************************************

   This function will set the background color to the value passed.

*************************************************************************/

backgnd(color)
int color;
{

        color &= 0x0F;
        inregs.ax = 0x0B00;                      /* Call BIOS to set backgn
d  */
        inregs.bx = 0;                           /* backgnd number = 0
*/
        inregs.bx = inregs.bx + (0xff & color);   /* Color number
*/
        INT86(0x10,&inregs );
}


/*************************************************************************

          This function will put a dot at the X,Y position in the color
        specified by COL.

*************************************************************************/

wdot(x,y,col)
```

```
int x,y,col;
{
        inregs.ax = 0x0C00;                          /* Call BIOS to set backgnd
*/
        inregs.ax = inregs.ax + (0xff & col);  /* pixel color value          *
/
        inregs.dx = y;                               /* Y position
*/
        inregs.cx = x;                               /* X position
*/
        INT86(0x10,&inregs );
 }


/*****************************************************************************

          This function will put a character at the current location of the
         cursor on the specified page.

 *****************************************************************************/

wrchr(chr,color,page)
int chr,color,page;
{
        inregs.ax = 0x0900;                           /* Call BIOS to put char
*/
        inregs.ax = 0xff & chr ;                     /* set character
*/
        inregs.bx = color << 8;                       /* set color number
*/
        inregs.bx = inregs.bx + (0xff & page);   /* set page #
*/
        inregs.cx = 1;                  /* number of times to write char   */

        INT86(0x10,&inregs );
 }


/*****************************************************************************


******************************************************************************/


/****************************************************************************

        This fft code performs a radix 4 fft.

        x - contains the real valued data
        y - contains the imaginary valued data
        n - is the number of points
        m - is the power to which the radix is raised

*****************************************************************************/
```

```
fft5(x,y,n,m)

int n,m;
double x[],y[];
{

int     n1, n2, i, j, k, i1,i2,i3;
double  a, e, c, s, xt, yt, co1,co2,co3,si1,si2,si3,b,r1,r2,r3,r4,s1,s2,s3,s4
;

        n2 = n;

        for (k=1; k <= m; k++)
         {
                n1 = n2;
                n2 = n2/4;
                e = 6.283185307179586 / n1;
                a = 0;

                for (j=1; j <= n2; j++)
                 {
                        b = a + a;
                        c = a + b;

                        /* Twiddle factor calculations */

                        co1 = cos(a);
                        co2 = cos(b);
                        co3 = cos(c);
                        si1 = sin(a);
                        si2 = sin(b);
                        si3 = sin(c);

                        a = j * e;

                        for (i=j; i <= n; i += n1)
                         {
                                /* radix four butterflies */

                                i1 = i  + n2;
                                i2 = i1 + n2;
                                i3 = i2 + n2;

                                r1 = x[i-1] + x[i2-1];
                                r3 = x[i-1] - x[i2-1];
                                s1 = y[i-1] + y[i2-1];
                                s3 = y[i-1] - y[i2-1];

                                r2 = x[i1-1] + x[i3-1];
                                r4 = x[i1-1] - x[i3-1];
                                s2 = y[i1-1] + y[i3-1];
                                s4 = y[i1-1] - y[i3-1];

                                x[i-1] = r1 + r2;
                                r2    = r1 - r2;
                                r1    = r3 - s4;
                                r3    = r3 + s4;
```

```
                                        y[i-1] = s1 + s2;
                                        s2    = s1 - s2;
                                        s1    = s3 + r4;
                                        s3    = s3 - r4;


                                        /* Twiddle factor multiplications */

                                        x[i1-1] = co1 * r3 + si1 * s3;
                                        y[i1-1] = co1 * s3 - si1 * r3;
                                        x[i2-1] = co2 * r2 + si2 * s2;
                                        y[i2-1] = co2 * s2 - si2 * r2;
                                        x[i3-1] = co3 * r1 + si3 * s1;
                                        y[i3-1] = co3 * s1 - si3 * r1;

                                 }

                         }

                 }

/*               descramble the ordering of the data                    */

        j = 1;
        n1 = n - 1;

        for (j=1; i <= n1; i++)
         {
                 if (i < j)
                  {
                         xt      = x[j-1];
                         x[j-1]  = x[i-1];
                         x[i-1]  = xt;

                         xt      = y[j-1];
                         y[j-1]  = y[i-1];
                         y[i-1]  = xt;
                  }
                 k = n/4;

                 while ((k*3) < j)
                  {
                         j = j - k*3;
                         k = k/4;
                  }

                 j = j + k;
         }

}
```

```
#include "math.h"
#include "color.h"
#include "stdio.h"

#define PAGE 0

extern float record_time;

float data_a[50000],data_b[50000]; /* 2 arrays, each 50 msec of data */
double x[16384], y[16384];                        /* arrays for 16k point fft
*/

/**************************************************************************

          This routine allows for the plotting, manipulation, and triggering
        of recieved data

          Entered by choosing DATA to SCREEN option from the data transfer
        section of the Board OS    .

**************************************************************************/

plot_data() {

int       plot_mode,start_time,stop_time;
float     lower_y_value,upper_y_value;
int       answer,i,mode;
int       color,style;
int       m,n,offset,signal;
float     delta_F;

  vmode(16);    /* Initalize screen to MODE 16  */

  start_time = 0;
  stop_time = 0;
  upper_y_value = 0;
  lower_y_value = 0;
  plot_mode = 1;

  answer = 1;                /* print options and current parm values */
  while(answer >=0)
   {

        cprintf(17,22,RED,RED,PAGE,"                  ");
        cprintf(17,1,RED,RED,PAGE,"[0]  :Starting time [ %d ]",start_time);
        cprintf(18,22,RED,RED,PAGE,"                  ");
        cprintf(18,1,RED,RED,PAGE,"[1]  :Stopping  time [ %d ]",stop_time);
        cprintf(19,22,RED,RED,PAGE,"                  ");
        cprintf(19,1,RED,RED,PAGE,"[2]  :Lower Y value [ %4.2f ]",lower_y_val
ue);
        cprintf(20,22,RED,RED,PAGE,"                  ");
        cprintf(20,1,RED,RED,PAGE,"[3]  :Upper  Y value [ %4.2f ]",upper_y_va
lue);
        cprintf(21,1,RED,RED,PAGE,"[4]  :Plot AXIS    ");

        cprintf(17,35,RED,RED,PAGE,"[10] :Get data in A");
        cprintf(18,35,RED,RED,PAGE,"[11] :Get data in B");
        cprintf(19,35,RED,RED,PAGE,"[12] :Plot Data   A");
```

```
    cprintf(20,35,RED,RED,PAGE,"[13] :Plot Data   B");
    cprintf(21,35,RED,RED,PAGE,"[14] :Auto fft menu");


    cprintf(17,60,RED,RED,PAGE,"[20] :Clear screen ");
    cprintf(18,60,RED,RED,PAGE,"[21] :Trans/Recieve");
    cprintf(19,75,RED,RED,PAGE,"    ");
    cprintf(19,60,RED,RED,PAGE,"[22] :Plot mode[ %d ]",plot_mode);
    cprintf(20,60,RED,RED,PAGE,"[23] :fft A->B      ");
    cprintf(21,60,RED,RED,PAGE,"[24] :EXIT          ");

  cprintf(22,1,RED,RED,PAGE,"Enter number of action to take :");
  scanf("%d",&answer);
  clear_line(24);


  /*  perform action based on action number entered on query */

  switch(answer)
   {

     case 0:
      {
             cprintf(22,1,RED,RED,PAGE,"Enter Starting Time                 :")
;
             scanf("%d",&start_time);
             clear_line(24);
             break;
      }

     case 1:
      {
             cprintf(22,1,RED,RED,PAGE,"Enter Stoping Time                  :")
;
             scanf("%d",&stop_time);
             clear_line(24);
             break;
      }

     case 2:
      {
             cprintf(22,1,RED,RED,PAGE,"Enter Lower Y value                 :")
;
             scanf("%f",&lower_y_value);
             clear_line(24);
             break;
      }

     case 3:
      {
             cprintf(22,1,RED,RED,PAGE,"Enter Upper Y value                 :")
;
             scanf("%f",&upper_y_value);
             clear_line(24);
             break;
      }

     case 4:
```

```
         {
                 cprintf(22,1,RED,RED,PAGE,"Plotting AXIS ");
                 style = 0;mode = 1;color = BLUE;
                 axis("X-axis","MAG",(float)start_time,(float)stop_time,lower_
y_value,upper_y_value,1,0,79,16,color);
                 clear_line(24);
                 break;
         }

        case 10:
         {
                 cprintf(22,1,RED,RED,PAGE,"Getting NEW Data in 'A' ");

                 board_to_array ( data_a );

                 clear_line(24);
                 break;
         }

        case 11:
         {
                 cprintf(22,1,RED,RED,PAGE,"Getting NEW Data in 'B' ");

                 board_to_array ( data_b );

                 clear_line(24);
                 break;
         }

        case 12:
         {
                 cprintf(22,1,RED,RED,PAGE,"Plot Data 'A', what color      :")
;
                 scanf("%d",&color);
                 clear_line(24);
                 style = 1;

                 if (plot_mode == 1)
                         mode = 1;
                 else
                         mode  = 2;

                 plot(data_a,start_time,stop_time,style,color,mode,lower_y_val
ue,upper_y_value);
                 break;
         }

        case 13:
         {
                 cprintf(22,1,RED,RED,PAGE,"Plot Data 'B', what color      :")
;
                 scanf("%d",&color);
                 clear_line(24);
                 style = 1;

                 if (plot_mode == 1)
                         mode = 1;
                 else
```

```
                    mode  = 2;

                plot(data_b,start_time,stop_time,style,color,mode,lower_y_val
ue,upper_y_value);
                break;
            }

        case 14:
            {
                /*    Initialize everything for auto fft action */

                vmode(16);

                cprintf(22,1,RED,RED,PAGE,"Enter starting color:");
                scanf("%d",&color);
                clear_line(24);
                style = 1;

                cprintf(22,1,RED,RED,PAGE,"Enter starting point:");
                scanf("%d",&offset);
                clear_line(24);

                cprintf(22,1,RED,RED,PAGE,"Enter power for Radix four fft (i.
e. 4**val: 4**6 = 4096:");
                scanf("%d",&m);
                clear_line(24);

                n=1;
                for (i=0; i<m; i++) n *= 4;

                if (plot_mode == 1)
                    mode = 1;
                else
                    mode  = 2;

                signal = 1;
                style = 0;color = BLUE;
                axis("FREQUENCY","MAG",(float)start_time,(float)stop_time,low
er_y_value,upper_y_value,1,0,79,16,color);


                /* Now do the auto stuff until we desire to quit  */

                while (signal)
                 {
                    trans_rec();

                    for (i=0; i<1000000; i++) i=i; /* delay */

                    board_to_array ( data_a );

                    for(i=0; i<n; i++) { x[i] = data_a[i+offset]; y[i] =
0.0; }

                    fft5(x,y,n,m);

                    for (i=0; i<n; i++)
                        data_b[i] = sqrt( x[i] * x[i] + y[i] * y[i]);
```

```
                              color++; if (color % 16 == 0) color++;

                              plot(data_b,start_time,stop_time,style,color,mode,low
er_y_value,upper_y_value);

                              cprintf(22,1,RED,RED,PAGE,"Enter 1 to keep going, 0 t
o stop:");

                              scanf("%d",&signal);
                              clear_line(24);
                        }

                  clear_line(24);
                  break;
            }

       case 20:
            {
                  vmode(16);    /* Initalize screen to MODE 16  */
                  break;
            }

       case 21:
            {
                  cprintf(22,1,RED,RED,PAGE,"Initiation of transmit/recieve ope
ration.");

                  trans_rec();
                  clear_line(24);
                  break;
            }

       case 22:
            {
                  cprintf(22,1,RED,RED,PAGE,"Plot mode (1- auto scale. 2- absol
ute scale) :");
                  scanf("%d",&plot_mode);

                  clear_line(24);
                  break;
            }

       case 23:
            {     /* Do fft of specified A data into B array */

                  cprintf(22,1,RED,RED,PAGE,"Enter starting point:");
                  scanf("%d",&offset);
                  clear_line(24);

                  cprintf(22,1,RED,RED,PAGE,"Enter power for Radix four fft (i.
e. 4**val; 4**6 = 4096:");
                  scanf("%d",&m);
                  clear_line(24);

                  n=1;
                  for (i=0; i<m; i++) n *= 4;

                  delta_f = 1000000.0 / n;
```

```
                for(i=0; i<n; i++) { x[i] = data_a[i+offset]; y[i] = 0.0; }

                fft5(x,y,n,m);

                for (i=0; i<n; i++)
                        data_b[i] = sqrt( x[i] * x[i] + y[i] * y[i]);

                clear_line(24);
                break;
            }

        case 24:
            {
                answer = -1;
                clear_line(24);
                break;
            }

        default :
            {
                break;
            }

      }

    }

}

/****************************************************************************
*

****************************************************************************
/

clear_line(row)
int row;
{
if((row > 24) || (row< 0))return(0);
printf(22,1,RED,RED,PAGE,"
");
return(0);
}
```

```
#include "stdio.h"
#define PI 3.14159265

#define TRUE       1
#define FALSE      0
#define max(a,b) ((a)>(b))?(a):(b)
#define abs(x)    ((x)<0?(-(x)):(x))
#define sign(a)  ((a)>0?1:((a)==0?0:(-1)))

struct REGS {
   short ax;
   short flags;
   short bx;
   short cx;
   short dx;
   short si;
   short di;
   short ds;
   short es;
}  inregs;


/**********************************************************************
 **********************************************************************
*
* Name            grline -- Draw a colored line
*
* Synopsis         npts = grline(pstart,pend,color);
*
*                 int npts            The number of points plotted is returned
*                 PT *pstart          Pointer to starting point of line
*                 PT *pend            Pointer to ending point of line
*                 int color           Color of the line; chosen from the
*                                    currently set palette.
*
* Description      This function draws a line on the current display page
*                 from *pstart to *pend using the specified color.  The
*                 actual number of points plotted to draw the line is
*                 returned.  If *pstart and *pend represent the same
*                 location, only one point is plotted.
*
* Method           The variables x and y keep count of when the tracing
*                 point coordinates whould be incremented.  The variable
*                 fplot is used as a flag when a new point is plotted.
*                 The same point is not plotted twice, because a color
*                 value greater than 128 should XOR the current point.
*
* Returns          npts                Number of points plotted to trace the line
.
*
* Version          3.0 (C)Copyright Blaise Computing Inc.      1983, 1984, 1986
*

**********************************************************************/

int grline(xo,yo,xm,ym,color)
int xo,yo,xm,ym;
int color;
```

```
{
    int deltax,deltay,steps,i;
    int incx,incy,x,y,fplot,npts;
    int xx,yy;

    /* First set up the increments and determine how many points must  */
    /* be plotted to trace the line.                                     */

    deltax = xm-xo;
    deltay = ym-yo;
    incx   = abs(deltax);
    incy   = abs(deltay);
    steps  = max(incx,incy);

    /* Initialize the counting variable, plot the first point and     */
    /* trace the rest of the line.                              */

    x      = 0;
    y      = 0;
    xx = xo;
    yy = yo;

    npts = 1;
    wtdot(xo,yo,color);

    for (i = 0; i <= steps; i++)
    {
        fplot = FALSE;
        x += incx;
        y += incy;
        if (y > steps)
        {
            y      -= steps;
            yy += sign(deltay);
            fplot  = TRUE;
        }
        if (x > steps)
        {
            x      -= steps;
            xx += sign(deltax);
            fplot  = TRUE;
        }
        if (fplot)                          /* Only plot if a new point      */
        {
            wtdot(xx,yy,color);
            npts++;
        }
    }

    return(npts);
}

/*****************************************************************************
/
```

```
# nclude "stdio.h"

#define JUST_RIGHT  1
#define JUST_LEFT   2
#define JUST_CENTER 3


int XO,YO,XM,YM;
float XINC,YINC,XRANGE,YRANGE;
float maxed(),mined();
float YOFFSET,YMAX,YMIN;


#define          TICS                    10
#define          VIDEO_PAGE               0
#define          X_PIXEL_SIZE             8
#define          Y_PIXEL_SIZE             14
#define          TIC_LENGTH               5
#define          Y_AXIS_TITLE_BLOCK_SIZE  3
#define          X_AXIS_TITLE_BLOCK_SIZE  1

/*      USE THESE ONLY IF YOU WANT CONTROL OF THE GLOABAL VARIABLES
extern  int XO,YO,XM,YM;
extern  float XINC,YINC,XRANGE,YRANGE;
extern  float points[501];
extern  float YOFFSET,YMAX,YMIN;
*/

float maxed(),mined();

/***************************************************************************
 ***************************************************************************

AXIS(XTITLE,YTITLE,XMIN,XMAX,YMIN,YMAX,XORIG,YORIG,XMAX,YMAX,COLOR)

Axis is a function that will print an AXIS at the position designated by
the row and column numbers XORIG,YORIG (upper left row/column) and XMAX,YMAX
(lower right row/column). The row/column is in CHARACTER format, in other
words, 1 < COL < 80, and 1 < ROW < 25 (this is from 80 characters across, and
25 lines of text).

YTITLE          char array       : title of Y axis.
XTITLE          char array       : title of X axis.
XMAX            float           : number printed on X axis far RIGHT
XMIN            float           : number printed on X axis far LEFT
YMAX            float           : number printed on Y Axis at  TOP
YMIN            float           : number printed on Y Axis at  BOTTOM
XORIG           int             : Upper right COLUMN value (1-79)
YORIG           int             : Upper right ROW value (1-24)
XMAX            int             : Bottom left COLUMN value (1-79)
XMIN            int             : Bottom left ROW value (1-24)
COLOR           int             : Valid COLOR from 0 to 15

AXIS does not return anything and provides no checking to see if the correct
values were passed. BE CAREFUL !!

EXAMPLE:
xo = 1    first column
```

```
yo = 1   first row
xm = 79  last column
yo = 12  half way down the screen

example    axis("X title ","Y title ",-10.0,10.0,-100.0,0.0,xo,yo,xm,ym,BLUE)
;

***************************************************************************/


axis(xtitle,ytitle,xmin,xmax,ymin,ymax,xo,yo,xm,ym,col)
float  xmin,xmax,ymin,ymax;
int    xo,yo,xm,ym,col;
char   xtitle[],ytitle[];
{
int i,xoel,ydel;
float xinc,yinc;
char c[10],string[80];


    xo = xo * X_PIXEL_SIZE;
    xm = xm * X_PIXEL_SIZE;
    yo = yo * Y_PIXEL_SIZE;
    ym = ym * Y_PIXEL_SIZE;


/* Setup GLOBAL variables   */

    XO = xo+(Y_AXIS_TITLE_BLOCK_SIZE*X_PIXEL_SIZE)+2;                /* leav
e space for one character + 2 */
    YO = yo;
    XM = xm;
    YM = ym-(X_AXIS_TITLE_BLOCK_SIZE*Y_PIXEL_SIZE)-2; /* leave space for two
characters +2 */


/* Print the X and Y titles   */

    stpjust(string,xtitle,' ',(XM/X_PIXEL_SIZE-XO/X_PIXEL_SIZE),JUST_CENTER)
;
    cprintf((ym/Y_PIXEL_SIZE-X_AXIS_TITLE_BLOCK_SIZE)-1,XO/X_PIXEL_SIZE+1,co
l,1,0,string); /* XTITLE PRINTED  */


    stpjust(string,ytitle,' ',((YM/Y_PIXEL_SIZE-2)-(YO/Y_PIXEL_SIZE+2)),JUST
_CENTER);
    c[1] = 0;                                      /* YTITLE PRINTED  */
    for(i=0;(i<( ((YM/Y_PIXEL_SIZE-1)-(YO/Y_PIXEL_SIZE+2)) ) && (string[i] !
=0));i++) {
        c[0] = string[i];
        cprintf((yo/Y_PIXEL_SIZE+i+1),xo/X_PIXEL_SIZE+2,col,1,0,c);

    }


/* Print the X and Y max/min values */
```

```
	cprintf((ym/Y_PIXEL_SIZE-X_AXIS_TITLE_BLOCK_SIZE)-1,XO/X_PIXEL_SIZE,col,
col,VIDEO_PAGE,"%6.2f",xmin);
	cprintf((ym/Y_PIXEL_SIZE-X_AXIS_TITLE_BLOCK_SIZE)-1,(XM/X_PIXEL_SIZE-6),
col,col,VIDEO_PAGE,"%6.2f",xmax);
	cprintf((yo/Y_PIXEL_SIZE)-1,xo/X_PIXEL_SIZE+1,col,col,VIDEO_PAGE,"%2.1f"
,ymax);
	cprintf((YM/Y_PIXEL_SIZE)-1,xo/X_PIXEL_SIZE+',col,col,VIDEO_PAGE,"%2.1f"
,ymin);

/* Draw BOXES  */

	grline(XO,YO,XM,YO,col);	/* BOX FOR GRAPH...  */
	grline(XM,YO,XM,YM,col);
	grline(XM,YM,XO,YM,col);
	grline(XO,YM,XO,YO,col);

	grline(xo-2,yo,XO-1,yo,col);	/* BOX LEFT OF GRAPH  */
	grline(XO-1,yo,XO-1,YM,col);
	grline(XO-1,YM,xo-2,YM,col);
	grline(xo-2,YM,xo-2,yo,col);

	grline(xo-2,YM+1,XM,YM+1,col);		/*  BOX ON BOTTOM OF GRAPH
*/
	grline(XM,YM+1,XM,ym+1,col);
	grline(XM,ym+1,xo-2,ym+1,col);
	grline(xo-2,ym+1,xo-2,YM+1,col);


/* Draw the TICK MARKS on all axises */

	yinc = (YM-YO)/(TICS*1.0);		/* LEFT TICK MARKS   */
	for(i=1;i<=TICS;i++) {
	   ydel = i*yinc + YO;
	   grline(XO, del,XO+TIC_LENGTH,ydel,col);
	}

	xinc = (XM-XO)/(TICS*1.0);		/* TOP TICK MARKS   */
	for(i=1;i<=TICS;i++) {
	   xdel = i*xinc + XO;
	   grline(xdel,YM,xdel,YM-TIC_LENGTH,col);
	}

	yinc = (YM-YO)/(TICS*1.0);		/* RIGHT TICK MARKS   */
	for(i=1;i<=TICS;i++) {
	   ydel = i*yinc + YO;
	   grline(XM,ydel,XM-TIC_LENGTH,ydel,col);
	}

	xinc = (XM-XO)/(TICS*1.0);		/* BOTTOM TICK MARKS   */
	for(i=1;i<=TICS;i++) {
	   xdel = i*xinc + XO;
	   grline(xdel,YO,xdel,YO+TIC_LENGTH,col);
	}


	return(0);
```

```
}

/*************************************************************************
/
/*************************************************************************
/

float maxed(data,pstart,pstop)
float data[];
int pstart,pstop;
{
int i;
float temp;

    temp = data[pstart];
    for(i=pstart;i<=pstop;i++) {
       if(temp < data[i])temp = data[i];
    }
    return(temp);

}

float mined(data,pstart,pstop)
float data[];
int pstart,pstop;
{
int i;
float temp;

    temp = data[pstart];
    for(i=pstart;i<=pstop;i++) {
       if(temp > data[i])temp = data[i];
    }
    return(temp);

}

/*************************************************************************
/
/*************************************************************************
```

PLOT(DATA,XSTART,XSTOP,STYLE,COLOR,MODE,YSTART,YSTOP)

```
   *  RETURNS :>  1 if successful
                  0 if unsuccessful
```

Plot will plot the data in the array DATA starting from the XSTART point and ending on the XSTOP that were passed. The Style variable selects one of the following styles:

```
      stype = 0            ;Normal lines with out zero cross lines
      style = 1            ;Normal lines with zero cross lines
      style = 2            ;Points only
      style = 3            ;Dashed lines
      style = 4            ;Small boxes    2 pixels wide
      style = 5            ;Medimum boxes  4 pixels wide
      style = 6            ;Large boxes     6 pixels wide
```

The COLOR passed is the color that is used to plot the data. The MODE variable selects one of the following plot modes:

```
        mode = 1                      ;Auto scale data to fit the graph.
                                        and ignor the passed ystart, and
                                       ystop variables.
        mode = 2                      ;Use the ystop and ystart variables to
                                       determin plotting scale.
```

The YSTART and YSTOP variables are used only when in MODE 2 of the plotting. In MODE 2 case, they represent the MIN/MAX of the Y plotting axis. This is not the same as the AXIS variable.

NOTE. This function can be in MODE 1 with out passing the YSTART and YSTOP variables.

```
*******************************************************************************
/


plot(data,xstart,xstop,style,color,mode,ystart,ystop)
float data[],ystart,ystop;
int xstart,xstop,color,mode,style;
{
int i,j,temp;
float y_num_points,x_num_points;
float ymin,ymax;
int x1,y1,x2,y2;


/* SCALE THE WAVEFORM  */

    ymin = ystart;
    ymax = ystop;

    XRANGE = XM-XO;
    YRANGE = YM-YO;
    x_num_points = xstop-xstart;
    XINC    = XRANGE/x_num_points;

    if(mode == 1)   {           /* Auto Scale       */
        ymax = maxed(data,xstart,xstop);
        ymin = mined(data,xstart,xstop);
    }
    if(mode == 2);        /* Absolute Scale   */

    y_num_points = ymax-ymin;
    YINC  = YRANGE/y_num_points;
    YMAX = ymax;
    YMIN = ymin;
    YOFFSET = - YINC * YMIN;


/* print the corss hairs for the zero lines  */

    if(style == 1) {
       temp = (YM - YOFFSET);
       drline(XO,temp,XM,temp,color);
```

```
    }

/*  PLOT THE WAVE FORM    */

    x1 = X0;
    y1 = (YM - YINC * data[xstart]) - YOFFSET;

    j = 0;
    temp = YM-YOFFSET;
    for(i=xstart;i<=xstop;i++) {
       x2  = X0 + j++*XINC;
       y2  = temp - YINC*data[i];
       grline(x1,y1,x2,y2,color);
       x1=x2;
       y1=y2;
    }

    return(0);
}


/**
*
* Name            stpjust -- Justify a string within a field
*
* Synopsis         presult = stpjust(ptarget,psource,fill,fldsize,code);
*
*                 char *presult     Pointer to the resulting target string
*                 char *ptarget     Pointer to the target string, which must
*                                  be at least (fldsize + 1) bytes long.
*                 char *psource     Pointer to the source string
*                 char fill         Character to be used for padding
*                 int  fldsize      Size of field to be filled
*                 int  code         Type of justification:
*                                  JUST_LEFT, JUST_CENTER, or JUST_RIGHT.
*
* Description      This function justifies or centers a string within a
*                 field of a specified size, padding with a given
*                 character if necessary.  The resulting string is exactly
*                 fldsize characters long.
*
*                 If the source string has more than fldsize characters,
*                 it is truncated to fit into the target.  Characters from
*                 the left, center, or right portion of the source are
*                 used depending on whether left, center, or right
*                 justification is specified, respectively.
*
*                 If the source string is fewer than fldsize bytes long
*                 (not counting the trailing NUL ('\0')), the remaining
*                 space is filled with the fill character.  If left
*                 justification is specified, the filling takes place on
*                 the right; if right justification, on the left; if
*                 centering, on both sides.
*
*                 If code has an unknown value, left justification is
*                 performed.
```

```
*
* Returns        presult          Pointer to the altered target string.
*               *ptarget          The altered target string.
*
* Version        3.0 (C)Copyright Blaise Computing Inc.      1986
*
**/

stpjust(ptarget,psource,fill,fldsize,code)
register char *ptarget,*psource;
char          fill;
int           fldsize,code;
{
    register int diff,i;
    int             numleft;
    char            *savetarget = ptarget;

    if (fldsize < 0)
        fldsize = 0;
    if ((diff = ((int) strlen(psource)) - fldsize) >= 0)
    {                           /* Use only a portion of source           */
        switch (code)
        {
            case JUST_RIGHT:              /* Skip leftmost characters     */
                psource += diff;
                break;
            case JUST_CENTER:             /* Use center characters        */
                psource += diff / 2;
                break;
            case JUST_LEFT:               /* Use leftmost characters      */
            default:
                break;
        }
        while (fldsize--)
            *ptarget++ = *psource++;
    }
    else
    {                           /* There's extra space to fill         */
        diff = -diff;            /* diff is number of spaces to fill     */
        switch (code)
        {                       /* numleft = number of spaces on left   */
            case JUST_RIGHT:
                numleft = diff;
                break;
            case JUST_CENTER:
                numleft = diff / 2;
                break;
            case JUST_LEFT:
            default:
                numleft = 0;
                break;
        }
        for (i = numleft; i; i--)
            *ptarget++ = fill;       /* Add the fill chars on the left  */
        while (*psource)
            *ptarget++ = *psource++;   /* Copy the string itself       */
        for (i = diff - numleft; i; i--)
            *ptarget++ = fill;       /* Add the fill chars on the right */
```

```
    }
    *ptarget = '\0';
    return savetarget;
}

clearline(row)
int row;
{
cprintf(row,1,0,0,0,"
");
return(0);
}
```