Adaptive compressive gain and level dependent spectral shaping circuitry for a hearing aid include a microphone to produce an input signal and a plurality of channels connected to a common circuit output. Each channel includes a filter with a preset frequency response to receive the input signal and to produce a filtered signal, a channel amplifier to amplify the filtered signal to produce a channel output signal, a threshold register to establish a channel threshold level, and a gain circuit. The gain circuit increases the gain of the channel amplifier when the channel output signal falls below the channel threshold level and decreases the gain of the channel amplifier when the channel output signal rises above the channel threshold level. A transducer produces sound in response to the signal passed by the common circuit output.
OTHER PUBLICATIONS


Moore et al., “Practical and Theoretical Considerations in Designing and Implementing Automatic Gain Control (AGC) in Hearing Aids”, Quaderni di Audiolgia, No. 4, 1988, pp. 522–527.


FIG. 3

INPUT / OUTPUT CURVES AS A FUNCTION OF COMPRESSION RATIO

RATIO = 0
RATIO = 0.125
RATIO = 0.25
RATIO = 0.50
RATIO = 1
RATIO = 2

O  □  △  X  □  △

OUTPUT LEVEL (dB)

INPUT LEVEL (dB)
ADAPTIVE GAIN AND FILTERING CIRCUIT FOR A SOUND REPRODUCTION SYSTEM

GOVERNMENT SUPPORT

This invention was made with U.S. Government support under Veterans Administration Contracts VA KV 674-P-857 and VA KV 674-P-1736 and National Aeronautics and Space Administration (NASA) Research Grant No. NAG10-0040. The U.S. Government has certain rights in this invention.

BACKGROUND OF THE INVENTION

The present invention relates to adaptive compressive gain and level dependent spectral shaping circuitry for a sound reproduction system and, more particularly, to such circuitry for a hearing aid.

The ability to perceive speech and other sounds over a wide dynamic range is important for employment and daily activities. When a hearing impairment limits a person's dynamic range of perceptible sound, incoming sound falling outside of the person's dynamic range should be modified to fall within the limited dynamic range to be heard. Soft sounds fall outside the limited dynamic range of many hearing impairments and must be amplified above the person’s hearing threshold with a hearing aid to be heard. Loud sounds fall within the limited dynamic range of many hearing impairments and do not require a hearing aid or amplification to be heard. If the gain of the hearing aid is set high enough to enable perception of soft sounds, however, intermediate and loud sounds will be uncomfortably loud. Because speech recognition does not increase over that obtained at more comfortable levels, the hearing-impaired person will prefer a lower gain for the hearing aid. However, a lower gain reduces the likelihood that soft sounds will be amplified above the hearing threshold. Modifying the operation of a hearing aid to reproduce the incoming sound at a reduced dynamic range is referred to herein as compression.

It has also been found that the hearing-impaired prefer a hearing aid which varies the frequency response in addition to the gain as sound level increases. The hearing-impaired may prefer a first frequency response and a high gain for low sound levels, a second frequency response and an intermediate gain for intermediate sound levels, and a third frequency response and a low gain for high sound levels. This operation of a hearing aid to vary the frequency response and the gain as a function of the level of the incoming sound is referred to herein as "level dependent spectral shaping.”

In addition to amplifying and filtering incoming sound effectively, a practical ear-level hearing aid design must accommodate the power, size and microphone placement limitations dictated by current commercial hearing aid designs. While powerful digital signal processing techniques are available, they can require considerable space and power so that most are not suitable for use in an ear-level hearing aid. Accordingly, there is a need for a hearing aid that varies its gain and frequency response as a function of the level of incoming sound, i.e., that provides an adaptive compressive gain feature and a level dependent spectral shaping feature each of which operates using a modest number of computations, and thus allows for the customization of variable gain and variable filter parameters according to a user’s preferences.

SUMMARY OF THE INVENTION

Among the several objects of the present invention may be noted the provision of a circuit in which the gain is varied in response to the level of an incoming signal; the provision of a circuit in which the frequency response is varied in response to the level of an incoming signal; the provision of a circuit which adaptively compresses an incoming signal occurring over a wide dynamic range into a limited dynamic range according to a user’s preference; the provision of a circuit in which the gain and the frequency response are varied in response to the level of an incoming signal; and the provision of a circuit which is small in size and which has minimal power requirements for use in a hearing aid.

Generally, in one form the invention provides an adaptive compressing and filtering circuit having a plurality of channels connected to a common output. Each channel includes a filter with preset parameters to receive an input signal and to produce a filtered signal, a channel amplifier which responds to the filtered signal to produce a channel output signal, a threshold circuit to establish a channel threshold level for the channel output signal, and a gain circuit. The gain circuit responds to the channel output signal and the channel threshold level to increase the gain setting of the channel amplifier up to a predetermined limit when the channel output signal falls below the channel threshold level and to decrease the gain setting of the channel amplifier when the channel output signal rises above the channel threshold level. The channel output signals are combined to produce an adaptively compressed and filtered output signal. The circuit is particularly useful when incorporated in a hearing aid. The circuit would include a microphone to produce the input signal and a transducer to produce sound as a function of the adaptively compressed and filtered output signal. The circuit could also include a second amplifier in each channel which responds to the filtered signal to produce a second channel output signal. The hearing aid may additionally include a circuit for programming the gain setting of the second channel amplifier as a function of the gain setting of the first channel amplifier.

Another form of the invention is an adaptive gain amplifier circuit having an amplifier to receive an input signal in the audible frequency range and to produce an output signal. The circuit includes a threshold circuit to establish a threshold level for the output signal. The circuit further includes a gain circuit which responds to the output signal and the threshold level to increase the gain of the amplifier up to a predetermined limit in increments having a magnitude dp when the output signal falls below the threshold level and to decrease the gain of the amplifier in decrements having a magnitude dm when the output signal rises above the threshold level. The output signal is compressed as a function of the ratio of dm over dp to produce an adaptively compressed output signal. The circuit is particularly useful in a hearing aid. The circuit may include a microphone to produce the input signal and a transducer to produce sound as a function of the adaptively compressed output signal.

Still another form of the invention is a programmable compressive gain amplifier circuit having a first amplifier to receive an input signal in the audible frequency range and to...
produce an amplified signal. The circuit includes a threshold circuit to establish a threshold level for the amplified signal. The circuit further includes a gain circuit which responds to the amplified signal and the threshold level to increase the gain setting of the first amplifier when the amplified signal rises above the threshold level and to decrease the gain setting of the first amplifier when the amplified signal falls below the threshold level. The amplified signal is thereby compressed. The circuit also has a second amplifier to receive the input signal and to produce an output signal. The circuit also has a gain circuit to program the gain setting of the second amplifier as a function of the gain setting of the first amplifier. The output signal is programmably compressed. The circuit is useful in a hearing aid. The circuit may include a microphone to produce the input signal and a transducer to produce sound as a function of the programmably compressed output signal.

Still another form of the invention is an adaptive filtering circuit having a plurality of channels connected to a common output, each channel including a filter with preset parameters to receive an input signal in the audible frequency range to produce a filtered signal and an amplifier which responds to the filtered signal to produce a channel output signal. The circuit includes a second filter with preset parameters which responds to the input signal to produce a characteristic signal. The circuit further includes a detector which responds to the characteristic signal to produce a control signal. The time constant of the detector is programmable. The circuit also has a log circuit which responds to the detector to produce a log value representative of the control signal. The circuit also has a memory to store a preselected table of log values and gain values. The memory responds to the log circuit to select a gain value for each of the amplifiers in the channels as a function of the produced log value. Each of the amplifiers in the channels responds to the memory to separately vary the gain of the respective amplifier as a function of the respective selected gain value. The channel output signals are combined to produce an adaptively filtered output signal. The circuit is useful in a hearing aid. The circuit may include a microphone to produce the input signal and a transducer to produce sound as a function of the adaptively filtered output signal.

Yet still another form of the invention is an adaptive filtering circuit having a filter with variable parameters to receive an input signal in the audible frequency range and to produce an adaptively filtered signal. The circuit includes an amplifier to receive the adaptively filtered signal and to produce an adaptively filtered output signal. The circuit additionally has a detector to detect a characteristic of the input signal and a controller which responds to the detector to vary the parameters of the variable filter and to vary the gain of the amplifier as functions of the detected characteristic.

Other objects and features will be in part apparent and in part pointed out hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of an adaptive compressive gain circuit of the present invention.

FIG. 2 is a block diagram of an adaptive compressive gain circuit of the present invention wherein the compression ratio is programmable.

FIG. 3 is a graph showing the input/output curves for the circuit of FIG. 2 using compression ratios ranging from 0-2.

FIG. 4 shows a four channel level dependent spectral shaping circuit wherein the gain in each channel is adaptively compressed using the circuit of FIG. 1.

FIG. 5 shows a four channel level dependent spectral shaping circuit wherein the gain in each channel is adaptively compressed with a programmable compression ratio using the circuit of FIG. 2.

FIG. 6 shows a four channel level dependant spectral shaping circuit wherein the gain in each channel is adaptively varied with a level detector and a memory.

FIG. 7 shows a level dependant spectral shaping circuit wherein the gain of the amplifier and the parameters of the filters are adaptively varied with a level detector and a memory.

FIG. 8 shows a two channel version of the four channel circuit shown in FIG. 6.

FIG. 9 shows the output curves for the control lines leading from the memory of FIG. 8 for controlling the amplifiers of FIG. 8.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

An adaptive filtering circuit of the present invention as it would be embodied in a hearing aid is generally indicated at reference number 10 in FIG. 1. Circuit 10 has an input 12 which represents any conventional source of an input signal such as a microphone, signal processor, or the like. A microphone is shown by way of example in FIG. 1. Input 12 also includes an analog to digital converter (not shown) for analog input signals if circuit 10 is implemented with digital components. Likewise, input 12 includes a digital to analog converter (not shown) for digital input signals if circuit 10 is implemented with analog components.

Input 12 is connected by a line 14 to an amplifier 16. The gain of amplifier 16 is controlled via a line 18 by an amplifier 20. Amplifier 20 amplifies the value stored in a gain register 24 according to a predetermined gain setting stored in a gain register 22 to produce an output signal for controlling the gain of amplifier 16. The output signal of amplifier 16 is connected by a line 28 to a limiter 26. Limiter 26 peak clips the output signal from amplifier 16 to provide an adaptively clipped and compressed output signal at output 30 in accordance with the invention, as more fully described below. The output 30, as with all of the output terminals identified in the remaining Figs. below, may be connected to further signal processors or to drive transducer 32 of a hearing aid.

With respect to the remaining components in circuit 10, a comparator 32 monitors the output signal from amplifier 16 via line 28. Comparator 32 compares the level of said output with a threshold level stored in a register 34 and outputs a comparison signal via a line 36 to a multiplexer 38. When the level of the output signal of amplifier 16 exceeds the threshold level stored in register 34, comparator 32 outputs a high signal via line 36. When the level of the output of amplifier 16 falls below the threshold level stored in register 34, comparator 32 outputs a low signal via line 36. Multiplexer 38 is also connected to a register 40 which stores a magnitude dp and to a register 42 which stores a magnitude dm. When multiplexer 38 receives a high signal via line 36, multiplexer 38 outputs a positive value corresponding to dp via line 44. An adder 46 is connected via line 44 to multiplexer 38 and is connected via a line 54 to gain register 24. Adder 46 adds the value output by multiplexer 38 to the value stored in gain register 24 and outputs the sum via a line 48 to update gain register 24. The circuit components for updating gain register 24 are enabled in response to a predetermined portion of a timing sequence produced by
a clock 50. Gain register 24 is connected by a line 52 to amplifier 20. The values stored in registers 22 and 24 thereby control the gain of amplifier 20. The output signal from amplifier 20 is connected to amplifier 16 for increasing the gain of amplifier 16 up to predetermined limit when the output level from amplifier 16 falls below the threshold level stored in register 34 and for decreasing the gain of amplifier 16 when the output level from amplifier 16 rises above the threshold level stored in register 34.

In one preferred embodiment, gain register 24 is a 12 bit register. The six most significant bits are connected by line 52 to control the gain of amplifier 16. The six least significant bits are updated by adder 46 via line 48 during the enabling portion of the timing sequence from clock 50. The new values stored in the six least significant bits are passed back to adder 46 via line 54. Adder 46 updates the values by dm or dp under the control of multiplexer 38. When the six least significant bits overflow the first six bits of gain register 24, a carry bit is applied to the seventh bit of gain register 24, thereby incrementing the gain setting of amplifier 20 by one bit. Likewise, when the six least significant bits underflow the first six bits of gain register 24, the gain setting of amplifier 20 is decremented one bit. Because the magnitudes dp and dm are stored in log units, the gain of amplifier 16 is increased and decreased by a constant percentage. A one bit change in the six most significant bits gain register 24 corresponds to a gain change in amplifier 16 of approximately 0.1 dB. Accordingly, the six most significant bits in gain register 24 provide a range of 32 decibels over which the conditions of adaptive limiting occur.

The sizes of magnitudes dp and dm are small relative to the value corresponding to the six least significant bits in gain register 24. Accordingly, there must be a net contribution of positive values corresponding to dp in order to raise the six least significant bits to their full count, thereby incrementing the next most significant bit in gain register 24. Likewise, there must be a net contribution of negative values corresponding to dm in order for the six least significant bits in gain register 24 to decrement the next most significant bit in gain register 24. The increments and decrements are applied as fractional values to gain register 24 which provides an averaging process and reduces the variance of the mean of the gain of amplifier 16. Further, since a statistical average of the percent clipping is the objective, it is not necessary to examine each sample. If the signal from input 12 is in digital form, clock 50 can operate at a frequency well below the sampling frequency of the input signal. For example, the sampling frequency of the input signal is divided by 512 in setting the frequency for clock 50 in FIG. 1.

In operation, circuit 10 adaptively adjusts the channel gain of amplifier 16 so that a constant percentage clipping by 16 is achieved over a range of levels of the signal from input 12. Assuming the input signal follows a Laplacian distribution, it is modeled mathematically with the equation:

\[ p(x) = \frac{1}{\sqrt{2\pi} \sigma R} e^{-\frac{(x-R \sqrt{\sigma^2 R})^2}{2\sigma^2 R}} \]  
(1)

In equation (1), \( R \) represents the overall root means square signal level of speech. A variable \( F_L \) is now defined as the fraction of speech samples that fall outside of the limits \((L, -L)\). By integrating the Laplacian distribution over the intervals \((-\infty, -L)\) and \((L, +\infty)\), the following equation for \( F_L \) is derived:

\[ F_L = \frac{1}{2} \left[ 1 - \text{erf} \left( \frac{R \sqrt{\sigma^2 R}}{\sigma} \right) \right] \]  
(2)

As above, when a sample of the signal from input 12 is in the limit set by register 34, the gain setting in gain register 24 is reduced by dm. When a sample of the signal from input 12 is not in limit, the gain is increased by dp. Therefore, circuit 10 will adjust the gain of amplifier 16 until the following condition is met:

\[ (1 - F_L)dp = F_L dm \]  
(3)

After adaption, the following relationships are found:

\[ dp = F_L (dp + dm) \]  
(4)

\[ RL = \sqrt{\frac{\sigma^2}{R}} \ln (1 + dm/dp) \]  
(5)

Within the above equations, the ratio \( RL \) represents a compression factor established by the ratio \( dm/dp \). The percentage of samples that are clipped at \( \pm L \) is given by:

\[ \% \text{ clipping} = F_L \times 100 \]  
(6)

Table I gives typical values that have been found useful in a hearing aid. Column three is the "headroom" in decibels between the root mean square signal value of the input signal and limiting.

<table>
<thead>
<tr>
<th>dm/dp</th>
<th>RL</th>
<th>R/L in dB</th>
<th>% clipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1/16</td>
<td>23.3</td>
<td>27.4</td>
<td>94</td>
</tr>
<tr>
<td>1/8</td>
<td>12.0</td>
<td>21.6</td>
<td>89</td>
</tr>
<tr>
<td>1/4</td>
<td>6.3</td>
<td>16.0</td>
<td>80</td>
</tr>
<tr>
<td>1/2</td>
<td>3.5</td>
<td>10.0</td>
<td>77</td>
</tr>
<tr>
<td>1</td>
<td>2.04</td>
<td>6.2</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>1.29</td>
<td>2.2</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>.88</td>
<td>.11</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>.64</td>
<td>.38</td>
<td>11</td>
</tr>
<tr>
<td>16</td>
<td>.50</td>
<td>.60</td>
<td>6</td>
</tr>
<tr>
<td>32</td>
<td>.40</td>
<td>.79</td>
<td>3</td>
</tr>
</tbody>
</table>

In the above equations, the relationship, \( RL = G \sigma R \), applies where \( G \) represents the gain prior to limiting and \( \sigma \) represents the root mean square speech signal level of the input signal. When the signal level \( \sigma \) changes, circuit 10 will adapt to a new state such that \( RL = G \sigma R \) returns to the compression factor determined by dp and dm. The initial rate of adaption is determined from the following equation:

\[ \text{dbg} = \text{f}(1 - \text{erf}(\text{r}\sqrt{\text{sigma}})) \]  
(7)

In equation (7), \( f \) represents the clock rate of clock 50. The path followed by the gain (\( G \)) is determined by solving the following equations recursively:

\[ G = G_0 \text{exp}(\text{dbg}) \]  
(8)

Within equations (8) and (9), the attack and release times for circuit 10 are symmetric only for a compression factor (\( RL \)) of 2.04. The attack time corresponds to the reduction of gain in response to an increase in signal \( \sigma \). Release time...
corresponds to the increase in gain after the signal level $\sigma$ is reduced. For a compression factor setting of 12, the release time is much shorter than the attack time. For a compression factor setting of 0.64 and 0.50, the attack time is much shorter than the release time. These latter values are preferable for a hearing aid.

As seen above, the rate of adaptation depends on the magnitudes of $dp$ and $dm$ which are stored in registers 40 and 42. These 6-bit registers have a range from 1/128 dB to 63/128(dB). Therefore, at a sampling rate of 16 kHz from clock 50, the maximum slope of the adaptive gain function ranges from 125 dB/sec to 8000 dB/sec. For a step change of 32 dB, this corresponds to a typical range of time constant from 256 milliseconds to four milliseconds respectively. If $dm$ is set to zero, the adaptive compression feature is disabled.

FIG. 2 discloses a circuit 60 which has a number of common circuit elements with circuit 10 of FIG. 1. Such common elements have similar functions and have been marked with common reference numbers. In addition to circuit 10, however, circuit 60 of FIG. 2 provides for a programmable compression ratio. Circuit 60 has a gain control 66 which is connected to a register 62 by a line 64 and to gain register 24 by a line 68. Register 62 stores a compression factor. Gain control 66 takes the value stored in gain register 24 to the power of the compression ratio stored in register 62 and outputs said power gain value via a line 70 to an amplifier 72. Amplifier 72 combines the power gain value on line 70 with the gain value stored in a register 74 to produce an output gain on a line 76. An amplifier 80 receives the output gain via line 76 for controlling the gain of amplifier 78. Amplifier 78 amplifies the signal from input 12 accordingly. The output signal from amplifier 78 is peak clipped by a limiter 82 and supplied as an output signal for circuit 60 at an output 82 in accordance with the invention.

To summarize the operation of circuit 60, the input to limiter 82 is generated by amplifier 78 whose gain is programmably set as a power of the gain setting stored in gain register 24, while the input to comparator 32 continues to be generated as shown in circuit 10 of FIG. 1. Further, one of the many known functions other than the power function could be used for programmably setting the gain of amplifier 78.

The improvement in circuit 60 of FIG. 2 over circuit 10 of FIG. 1 is seen in FIG. 3 which shows the input/output curves for compression ratios ranging from zero through two. The curve corresponding to a compression ratio of one is the single input/output curve provided by circuit 10 in FIG. 1. Circuit 60 of FIG. 2, however, is capable of producing all of the input/output curves shown in FIG. 3.

In practice, circuit 10 of FIG. 1 or circuit 60 of FIG. 2 may be used in several parallel channels, each channel filtered to provide a different frequency response. Narrow band or broad band filters may be used to provide maximum flexibility in fitting the hearing aid to the patient's hearing deficiency. Broad band filters are used if the patient prefers one hearing aid characteristic at low input signal levels and another characteristic at high input signal levels. Broad band filters can also provide different spectral shaping depending on background noise level. The channels are preferably constructed in accordance with the filter/limit/filter structure disclosed in U.S. Pat. No. 5,111,419 (hereinafter "the '419 patent") and incorporated herein by reference.

FIG. 4 shows a 4-channel filter/limit/filter structure for circuit 10 of FIG. 1. While many types of filters can be used for the channel filters of FIG. 4 and the other Figs., FIR filters are the most desirable. Each of the filters F1, F2, F3 and F4 in FIG. 4 are symmetric FIR filters which are equal in length within each channel. This greatly reduces phase distortion in the channel output signals, even at band edges. The use of symmetric filters further requires only about one-half as many registers to store the filter coefficients for a channel, thus allowing a simpler circuit implementation and lower power consumption. Each channel response may be programmed to be a band-pass filter which is contiguous with adjacent channels. Therefore, filters F1-F4 constitute variable filters with separately varying filter parameters. In this mode, filters F1 through F4 have preset filter parameters for selectively passing input 12 over a predetermined range of audible frequencies while substantially attenuating any of input 12 not occurring in the predetermined range. Likewise, channel filters F1 through F4 can be programmed to be wide band to produce overlapping channels. In this mode, filters F1 through F4 have preset filter parameters for selectively altering input 12 over substantially all of the audible frequency range. Various combinations of these two cases are also possible. Since the filter coefficients are arbitrarily specified, in-band shaping is applied to the band-pass filters to achieve smoothly varying frequency gain functions across all four channels. An output 102 of a circuit 100 in FIG. 4 provides an adaptively compressed and filtered output signal comprising the sum of the filtered signals at outputs 30 in each of the four channels identified by filters F1 through F4.

FIG. 5 shows a four channel filter/limit/filter circuit 110 wherein each channel incorporates circuit 60 of FIG. 2. An output 112 in FIG. 5 provides a programmably compressed and filtered output signal comprising the sum of the filtered signals at outputs 82 in each of the four channels identified by filters F1 through F4.

The purpose of the adaptive gain factor in each channel of the circuitry of FIGS. 4 and 5 is to maintain a specified constant level of envelope compression over a range of inputs. By using adaptive compressive gain, the input/output function for each channel is programmed to include a linear range for which the signal envelope is unchanged, a higher input range over which the signal envelope is compressed by a specified amount, and the highest input range over which envelope compression increases as the input level increases. This adaptive compressive gain feature adds an important degree of control over mapping a widely dynamic input signal into the reduced auditory range of the impaired ear.

The design of adaptive compressive gain circuitry for a hearing aid presents a number of considerations, such as the wide dynamic range, noise pattern and bandwidth found in naturally occurring sounds. Input sounds present at the microphone of a hearing aid vary from quiet sounds (around 30 dB SPL) to those of a quiet office area (around 50 dB SPL) to much more intense transient sounds that may reach 120 dB SPL. Therefore, at a constant level of envelope compression increases as the input level increases. This adaptive compressive gain feature adds an important degree of control over mapping a widely dynamic input signal into the reduced auditory range of the impaired ear.

The design of adaptive compressive gain circuitry for a hearing aid presents a number of considerations, such as the wide dynamic range, noise pattern and bandwidth found in naturally occurring sounds. Input sounds present at the microphone of a hearing aid vary from quiet sounds (around 30 dB SPL) to those of a quiet office area (around 50 dB SPL) to much more intense transient sounds that may reach 120 dB SPL. Therefore, at a constant level of envelope compression increases as the input level increases. This adaptive compressive gain feature adds an important degree of control over mapping a widely dynamic input signal into the reduced auditory range of the impaired ear.
Electronic circuit noise and processing noise limit the quietest sounds that can be processed. A conventional hearing aid microphone has an equivalent input noise figure of 25 dB SPL, which is close to the estimated 20 dB noise figure of a normal ear. If this noise figure is used as a lower bound on the input dynamic range and 120 dB SPL is used as an upper bound, the input dynamic range of good hearing aid system is about 100 dB. Because the microphone will begin to saturate at 90 to 100 dB SPL, a lesser dynamic range of 75 dB is workable.

Signal bandwidth is another design consideration. Although it is possible to communicate over a system with a bandwidth of 3 kHz or less and it has been determined that 3 kHz carries most of the speech information, hearing aids with greater bandwidth result in better articulation scores. Skinner, M. W. and Miller, J. D., Amplification Bandwidth and Intelligibility of Speech in Quiet and Noise for Listeners with Sensorineural Hearing Loss, 22:253–79 Audiology (1983). Accordingly, the embodiment disclosed in FIG. 1 has a 6 kHz upper frequency cut-off.

The filter structure is another design consideration. The filters must achieve a high degree of versatility in programming bandwidth and spectral shaping to accommodate a wide range of hearing impairments. Further, it is desirable to use shorter filters to reduce circuit complexity and power consumption. It is also desirable to be able to increase filter gain for frequencies of reduced hearing sensitivity in order to improve signal audibility. However, studies have shown that a balance must be maintained between gain at low frequencies and gain at high frequencies. It is recommended that the gain difference across frequency should be no greater than 30 dB, Skinner, M. W., Hearing Aid Evaluation, Prentice Hall (1988). Further, psychometric functions often used to calculate a “prescriptive” filter characteristic are generally smooth, slowly changing functions of frequency that do not require a high degree of frequency resolution to fit.

Within the above considerations, it is preferable to use FIR filters with transition bands of 1000 Hz and out of band rejection of 40 dB. The required filter length is determined from the equation:

\[ L \approx \left(\frac{20 \log (10^{-1.8} \times 10^{-7})}{14.37 \log (10)}\right) + 1 \]  

(10)

In equation (10), L represents the number of filter taps, \( T \) represents the maximum error in achieving a target filter characteristic, \(-20 \log (10)\) represents the out of band rejection in decibals, \( T_B \) represents the transition band, and \( f_s \) is the sampling rate. See Kaiser, Nonrecursive Filter Design Using the \( L_o \)-SINH Window Function, Proc., IEEE Int. Symposium on Circuits and Systems (1974). For an out of band rejection figure of 35 dB with a transition band of 1000 Hz and a sampling frequency of 16 kHz, the filter must be approximately 31 taps long. If a lower out of band rejection of 30 dB is acceptable, the filter length is reduced to 25 taps. This range of filter lengths is consistent with the modest filter structure and low power limitations of a hearing aid.

All of the circuits shown in FIGS. 1 through 9 use log encoded data. See the ‘419 patent. Log encoding is similar to u-law and A-law encoding used in Codexcs and has the same advantages of extending the dynamic range, thereby making it possible to reduce the noise floor of the system as compared to linear encoding. Log encoding offers the additional advantage that arithmetic operations are performed directly on the log encoded data. The log encoded data are represented in the hearing aid as a sign and magnitude as follows:

\[ x = \text{sign}(y) \log(10) \log(|y|) \]  

(11)

In equation (11), \( B \) represents the log base, which is positive and close to but less than unity, \( x \) represents the log value and \( y \) represents the equivalent linear value. A reciprocal relation for \( y \) as a function of \( x \) follows:

\[ y = \text{sign}(x) B^x \]  

(12)

If \( x \) is represented as sign and an 8-bit magnitude and the log base is 0.941, the range of \( y \) is \( \pm 1 \times 10^{-7} \). This corresponds to a dynamic range of 134 dB. The general expression for dynamic range as a function of the log base \( B \) and the number of bits used to represent the log magnitude Value N follows:

\[ \text{dynamic range (dB)} = 20 \log (10) \left( B^N \right) \]  

(13)

An advantage of log encoding over u-law encoding is that arithmetic operations are performed directly on the encoded signal without conversion to another form. The basic FIR filter equation, \( y(n) = 2^a x(n-1) \), is implemented recursively as a succession of add and table lookup operations in the log domain. Multiplication is accomplished by adding the magnitude of the operands and determining the sign of the result. The sign of the result is a simple exclusive-or operation on the sign bits of the operands. Addition (and subtraction) are accomplished in the log domain by operations of subtraction, table lookup, and addition. Therefore, the sequence of operations required to form the partial sum of products of the FIR filter in the log domain are addition, subtraction, table lookup, and addition.

Addition and subtraction in the log domain are implemented by using a table lookup approach with a sparsely populated set of tables \( T_0, T_1 \) stored in a memory (not shown). Adding two values, \( x \) and \( y \), is accomplished by taking the ratio of the smaller magnitude to the larger and adding the value from the log table \( T_0 \) to the smaller. Subtraction is similar and uses the log table \( T_1 \). Since \( x \) and \( y \) are in log units, the ratio, \( \log(x/y) \), is used which is made available by using the log table \( T_1 \) or \( T_0 \) to use is determined by an exclusive-or operation on the sign bits of \( x \) and \( y \). Whether the table value is added to \( x \) or to \( y \) is determined by subtracting \( \log(1) \) from \( \log(1) \) and testing the sign bit of the result.

Arithmetic roundoff errors in using log values for multiplication are not significant. With an 8-bit representation, the log magnitude values are restricted to the range 0 to 255. Zero corresponds to the largest possible signal value and 255 to the smallest possible signal value. Log values less than zero cannot occur. Therefore, overflow can only occur for the smallest signal values. Product log values greater than 255 are truncated to 255. This corresponds to a smallest signal value (255 LU’s) that is 134 dB smaller than the maximum signal value. Therefore, if the system is scaled by setting the amplifier gains so that 0 LU corresponds to 130 dB SPL, the truncation errors of multiplication (255 LU) correspond to -134 dB relative to the maximum possible signal value (0 LU). In absolute terms, this provides a -4 dB SPL or -43 dB SPL spectrum level, which is well below the normal hearing threshold.

Roundoff errors of addition and subtraction are much more significant. For example, adding two numbers of equal magnitude together results in a table lookup error of 2.4%.
Conversely, adding two values that differ by three orders of magnitude results in an error of 0.1%. The two tables, $T_c$ and $T_\infty$, are sparsely populated. For a log base of 0.941 and table values represented as an 8-bit magnitude, each table contains 57 nonzero values. If it is assumed that the errors are uniformly distributed (that each table value is used equally often on the average), then the overall average error associated with table roundoff is 1.01% for $T_c$ and 1.02% for $T_\infty$.

Table errors are reduced by using a log base closer to unity and a greater number of bits to represent log magnitude. However, the size of the table grows and quickly becomes impractical to implement. A compromise solution for reducing error is to increase the precision of the table entries without increasing the table size. The number of nonzero entries increases somewhat. Therefore, in implementing the table lookup in the digital processor, two additional bits of precision are added to the table values. This is equivalent to using a temporary log base which is the fourth root of 0.941 (0.985) for calculating the FIR filter summation. The change in log base increases the number of nonzero entries in each of the tables by 22, but reduces the average error by a factor of four. This increases the output SNR of a given filter by 12 dB. The $T_c$ and $T_\infty$ tables are still sparsely populated and implemented efficiently in VLSI form.

In calculating the FIR equation, the table lookup operation is applied recursively $N-1$ times, where $N$ is the order of the filter. Therefore, the total error that results is greater than the average table roundoff error and a function of filter order. If it is assumed that the errors are uniformly distributed and that the input signal is white, the expression for signal to roundoff noise ratio follows:

$$
\sigma^2_{\text{SNR}} = \frac{\sigma^2_{\text{SNR}}}{\sigma^2_{\text{SNR}}} = \left(\sum \text{log entries}\right) (N-1) (10)^{\log_{10}(\sigma^2_{\text{SNR}})}
$$

In equation (14), $\sigma^2_{\text{SNR}}$ represents the noise variance at the output of the filter, $\sigma^2_{\text{SNR}}$ represents the signal variance at the output of the filter, and $\sigma^2_{\text{SNR}}$ represents the average percent table error. Accordingly, the filter noise is dependent on the table lookup error, the magnitude of the filter coefficients, and the order of summation. The coefficient used first introduces an error that is multiplied by $N-1$. The coefficient used second introduces an error that is multiplied by $N-2$ and so on. Since the error is proportional to coefficient magnitude and order of summation, it is possible to minimize the overall error by ordering the smallest coefficients earliest in the calculation. Since the end tap values for symmetric filters are generally smaller than the center tap value, the error was further reduced by calculating partial sums using coefficients from the outside toward the inside.

In FIGS. 4 and 5, FIR filters F1 through F4 represent channel filters which are divided into two cascaded parts. Limiters 26 and 80 are implemented as part of the log multiply operation. $G_1$ is a gain factor that, in the log domain, is subtracted from the samples at the output of the first FIR filter. If the sum of the magnitudes is less than zero (maximum signal value), it is clipped to zero. $G_2$ represents an attenuation factor that is added (in the log domain) to the clipped samples. $G_3$ is used to set the maximum output level of the channel.

Log quantizing noise is a constant percentage of signal level except for low input levels that are near the smallest quantizing steps of the encoder. Assuming a Laplacian signal distribution, the signal to quantizing noise ratio is given by the following equation:

$$
\text{SNR(dB)} = 10 \log_{10} \left( \frac{20 \log_{10}(N_\infty)}{20 \log_{10}(N_c)} \right)
$$

For a log base of 0.941, the SNR is 35 dB. The quantizing noise is white and, since equation (15) represents the total noise energy over a bandwidth of 8 kHz, the spectrum level is 39 dB less or 74 dB smaller than the signal level. The ear inherently masks the quantizing noise at this spectrum level. Schroeder, et al., Optimizing Digital Speech Coders by Exploiting Masking Properties of the Human Ear, Vol. 66(6) J. Acoust. Soc. Am. pp.1647–52 (December 1979). Thus, log encoding is ideally suited for auditory signal processing. It provides a wide dynamic range that encompasses the range of levels of naturally occurring signals, provides sufficient SNR that is consistent with the limitation of the ear to resolve small signals in the presence of large signals, and provides a significant savings with regard to hardware.

The goal of the fitting system is to program the digital hearing aid to achieve a target real-ear gain. The real-ear gain is the difference between the real-ear-aided-response (REAR) and the real-ear-unaided-response (REUR) as measured with and without the hearing aid on the patient. It is assumed that the target gain is specified by the audiologist or calculated from one of a variety of prescriptive formulae chosen by the audiologist that is based on audiometric measures. There is not a general consensus about which prescription is best. However, prescriptive formulae are generally quite simple and easy to implement on a small host computer. Various prescriptive fitting methods are discussed in Chapter 6 of Skinner, M. W., Hearing Aid Evaluation, Prentice Hall (1988).

Assuming that a target real-ear gain has been specified, the following strategy is used to automatically fit the four channel digital hearing aid where each channel is programmed as a band pass filter which is contiguous with adjacent channels. The real-ear measurement system disclosed in U.S. Pat. No. 4,548,082 (hereinafter "the '082 patent") and incorporated herein by reference is used. First, the patient’s REUR is measured to determine the patient’s normal, unoccluded ear canal resonance. Then the hearing aid is placed on the patient. Second, the receiver and earmold are calibrated. This is done by setting $G_2$ of each channel to maximum attenuation (−134 dB) and turning on the noise generator of the adaptive feedback equalization circuit shown in the '082 patent. This drives the output of the hearing aid with a flat-spectrum-level, pseudorandom noise sequence. The noise in the ear canal is then deconvolved with the pseudorandom sequence to obtain a measure of the output transfer characteristic ($H_1$) of the hearing aid. Third, the microphone is calibrated. This is done by setting the channels to a flat nominal gain of 20 dB. The cross-correlation of the sound in the ear canal with the reference sound then represents the overall transfer characteristic of the hearing aid and includes the occlusion of sound by the earmold. The microphone calibration (Hm) is computed by subtracting $H_1$ from this measurement. Last, the channel gain functions are specified and filter coefficients are computed using a window design method. See Rahiner and Schafer, Digital Processing of Speech Signals, Prentice Hall (1978). The coefficients are then downloaded in bit-serial order to the coefficient registers of the processor. The coefficient registers are connected together as a single serial shift register for the purpose of downloading and uploading values.

The channel gains are derived as follows. The acoustic gain for each channel of the hearing aid is given by:

$$
G_{\text{Gain}}(\text{dB}) = \frac{H_{\text{cal}}(\text{dB}) + H_{\text{cal}}(\text{dB}) + G_{\text{cal}}(\text{dB})}{G_{\text{cal}}(\text{dB})}
$$

The filter shape for each channel is determined by setting the gain in equation (16) to the desired real-ear gain plus the
open-ear resonance. Since $G_m$ and $G_n$ are gain constants for the channel and independent of frequency, they do not enter into the calculation at this point. The normalized filter characteristics is determined from the following equation.

$$H_n = 0.5 \times \text{(Desired Real-ear gain - open ear cal)} \times H_n + G_n$$ \hspace{1cm} (17)

$H_n$ and $H_n$ represent the microphone and receiver calibration measures, respectively, that were determined for the patient with the real ear measurement system and $G_n$ represents a normalization gain factor for the filter that is included in the computation of $G_m$ and $G_n$. $H_n$ and $H_n$ include the transducer transfer characteristics in addition to the frequency response of the amplifier and any signal conditioning filters. Once $H_n$ is determined, the maximum output of each channel, which is limited by $L_n$, are represented by $G_n$, as follows:

$$G_n = MPO \times L - \text{avg}(-H_n + H_n - G_n)$$ \hspace{1cm} (18)

In equation (18), the "avg" operator gives the average of filter gain and receiver sensitivity at filter design frequencies within the channel. $L$ represents a fixed level for all channels such that signals falling outside the range $-L$ are peaked-clipped at $-L$. $G_n$ represents the filter normalization gain, and $MPO$ represents the target maximum power output. Overall gain is then established by setting $G_m$, as follows:

$$G_m = 2G_n - G_{in}$$ \hspace{1cm} (19)

$G_n$ represents the gain normalization factor of the filters that were designed to provide the desired linear gain for the channel.

By using the above approach, target gains typically are realized to within 3 dB over a frequency range of from 100 Hz to 6000 Hz. The error between the step-wise approximation to the MPO function and the target MPO function is also small and is minimized by choosing appropriate crossover frequencies for the four channels.

Because the channel filters are arbitrarily specified, an alternative fitting strategy is to prescribe different frequency-gain shapes for signals of different levels. By choosing appropriate limit levels in each channel, a transition from the characteristics of one channel to the characteristics of the next channel will occur automatically as a function of signal level. For example, a transparent or low-gain function is used for high-level signals and a high-gain function is used for low-level signals. The adaptive gain feature in each channel provides a means for controlling the transition from one channel characteristic to the next. Because of recruitment and the way the impaired ear works, the gain functions are generally ordered from highest gain for soft sounds to the lowest gain for loud sounds. With respect to circuit 100 of FIG. 4, this is accomplished by setting $G_1$ in gain register 22 very high for the channel with the highest gain for the soft sounds. The settings for $G_1$ in gain registers 22 of the next succeeding channels are sequentially decreased, with the $G_1$ setting being unity in the last channel which channel has the lowest gain for loud sounds. A similar strategy is used for circuit 110 of FIG. 5, except that $G_1$ must be set in both gain registers 22 and 74. In this way, the channel gain settings in circuits 100 and 110 of FIGS. 4 and 5 are sequentially modified from first to last as a function of the level of input 12.

The fitting method is similar to that described above for the four-channel fitting strategy. Real-ear measurements are used to calibrate the ear, receiver, and microphone. However, the filters are designed differently. One of the channels is set to the lowest gain function and highest ACG threshold. Another channel is set to a higher-gain function, which adds to the lower-gain function and dominates the spectral shaping at signal levels below a lower ACG threshold setting for that channel. The remaining two channels are set to provide further gain contributions at successively lower signal levels. Since the channel filters are symmetric and equal length, the gains will add in the linear sense. Two channels set to the same gain function will provide 6 dB more gain than either channel alone. Therefore, the channels filters are designed as follows:

$$H_1 = \log_6 D_1$$ \hspace{1cm} (20)

$$H_2 = \log_6 (10^{50} - 10^{50})$$ \hspace{1cm} (21)

$$H_3 = \log_6 (10^{50} - 10^{50} - 10^{50})$$ \hspace{1cm} (22)

$$H_4 = \log_6 (10^{50} - 10^{50} - 10^{50} - 10^{50})$$ \hspace{1cm} (23)

where: $D_1 < D_2 < D_3 < D_4$. $D_n$ represents the filter design target in decibels that gives the desired insertion gain for the hearing aid and is derived from the desired gains specified by the audiologist and corrected for ear canal resonance and receiver and microphone calibrations as described previously for the four-channel fit. The factors $\delta$, in the above expressions takes into account that each channel has two filters in cascade.

The processor described above has been implemented in custom VLSI form. When operated at 5 volts and at a 16-kHz sampling rate, it consumes 4.6 mA. When operated at 3 volts and at the same sampling rate, it consumes 2.8 mA. When the circuit is implemented in a low-voltage form, it is expected to consume less than 1 mA when operated from a hearing aid battery. The processor has been incorporated into a bench-top prototype version of the digital hearing aid. Results of fitting hearing-impaired subjects with the processor described in FIGS. 12 through 15, the simplified circuitry of FIGS. 6 through 9 is used. In FIG. 6, a circuit 120 includes an input 12 which represents any conventional source of an input signal such as a microphone, signal processor, or the like. A microphone is shown by way of example. Input 12 also includes an analog to digital converter (not shown) for analog input signals if circuit 120 is implemented with digital components. Likewise, input 12 includes a digital to analog converter (not shown) for digital input signals if circuit 120 is implemented with analog components.

Input 12 is connected to a group of filters F1 through F4 and a filter S1 over a line 122. Filters F1 through F4 provide separate channels with filter parameters preset as described above for the multichannel circuits of FIGS. 4 and 5. Each of filters F1, F2, F3 and F4 outputs an adaptively filtered signal via a line 124, 126, 128 and 130 which is amplified by a respective amplifier 132, 134, 136 and 138. Amplifiers 132 through 138 each provide a channel output signal which is combined by a line 140 to provide an adaptively filtered signal at an output 142 of circuit 120.

Filter S1 has parameters which are set to extract relevant signal characteristics present in the input signal. The output of filter S1 is received by an envelope detector 144 which
detects said characteristics. Detector 144 preferably has a programmable time constant for varying the relevant period of detection. When detector 144 is implemented in analog form, it includes a full wave rectifier and a resistor/capacitor circuit (not shown). The resistor, the capacitor, or both, are variable for programming the time constant of detector 144. When detector 144 is implemented in digital form, it includes an exponentially shaped filter with a programmable time constant. In either event, the "on" time constant is shorter than the relatively long "off" time constant to prevent excessively loud sounds from existing in the output signal for extended periods.

The output of detector 144 is a control signal which is transformed to log encoded data by a log transformer 146 using standard techniques and as more fully described above. The log encoded data represents the extracted signal characteristics present in the signal at input 12. A memory 148 stores a table of signal characteristic values and related amplifier gain values in log form. Memory 148 receives the log encoded data from log transformer 146 and, in response thereto, recalls a gain value for each of amplifiers 132, 134, 136 and 138 as a function of the log value produced by log transformer 146. Memory 148 outputs the gain values via a set of lines 150, 152, 154 and 156 to amplifiers 132, 134, 136 and 138 for setting the gains of the amplifiers as a function of the gain values. Arbitrary overall gain control functions and blending of signals from each signal processing channel are implemented by changing the entries in memory 148.

In use, circuit 120 of FIG. 6 may include a greater or lesser number of filtered channels than the four shown in FIG. 6. Further, circuit 120 may include additional filters, detectors and log transformers corresponding to filter detector 144 and log transformer 146 for providing additional input signal characteristics to memory 148. Still further, any or all of the filtered signals in lines 124, 126, 128 or 130 could be used by a detector(s), such as detector 144, for detecting an input signal characteristic for use by memory 148.

FIG. 7 includes input 12 for supplying an input signal to a circuit 160. Input 12 is connected to a variable filter 162 and to a filter S1 via a line 164. Variable filter 162 provides an adaptively filtered signal which is amplified by an amplifier 166. A limiter 168 peak clips the adaptively filtered output signal of amplifier 166 to produce a limited output signal which is filtered by a variable filter 170. The adaptively filtered and clipped output signal of variable filter 170 is provided at output 171 of circuit 160.

Filter S1, a detector 144 and a log transformer 146 in FIG. 6 perform similar functions to the like numbered components found in FIG. 6. Memory 162 stores a table of signal characteristic values, related filter parameters, and related amplifier gain values in log form. Memory 162 responds to the input from log transformer 146 by recalling filter parameters and an amplifier gain value as functions of the log value produced by log transformer 146. Memory 162 outputs the recalled filter parameters via a line 172 and the recalled gain value via a line 174. Filters 162 and 170 receive said filter parameters via line 172 for setting the parameters of filters 162 and 170. Amplifier 166 receives said gain value via line 174 for setting the gain of amplifier 166. The filter coefficients are stored in memory 162 in sequential order of input signal level to control the selection of filter coefficients as a function of input level. Filters 162 and 170 are preferably FIR filters of the same construction and length and are set to the same parameters by memory 162. In operation, the circuit 160 is also used by taking the output signal from the output of amplifier 166 to achieve desirable results. Limiter 168 and variable filter 170 are shown, however, to illustrate the filter/limit/filter structure disclosed in the 419 patent in combination with the pair of variable filters 162 and 170.

With a suitable choice of filter coefficients, a variety of level dependent filtering is achieved. When memory 162 is a random-access memory, the filter coefficients are tailored to the patient's hearing impairment and stored in the memory from a host computer during the fitting session. The use of the host computer is more fully explained in the '082 patent.

A two channel version of circuit 120 in FIG. 6 is shown in FIG. 8 as circuit 180. Like components of the circuits in FIGS. 6 and 8 are identified with the same reference numerals. A host computer (such as the host computer disclosed in the '082 patent) is used for calculating the F1 and F2 filter coefficients for various spectral shaping, for calculating entries in memory 148 for various gain functions and blending functions, and for down-loading the values to the hearing aid.

The gain function for each channel is shown in FIG. 9. A segment "a" of a curve G1 provides a "voice switch" characteristic at low signal levels. A segment "b" provides a linear gain characteristic with a spectral characteristic determined by filter F1 in FIG. 8. A segment "c" and "d" provide a transition between the characteristics of filters F1 and F2. A segment "e" represents a linear gain characteristic with a spectral characteristic determined by filter F2. Lastly, segment "f" corresponds to a region over which the level of output 142 is constant and independent of the level of input 12.

The G1 and G2 functions are stored in a random access memory such as memory 148 in FIG. 8. The data stored in memory 148 is based on the specific hearing impairment of the patient. The data is derived from an appropriate algorithm in the host computer and down-loaded to the hearing aid model during the fitting session. The coefficients for filters F1 and F2 are derived from the patients residual hearing characteristic as follows: Filter F2, which determines the spectral shaping for loud sounds, is designed to match the patients UCL function. Filter F1, which determines the spectral shaping for softer sounds, is designed to match the patients MCL or threshold functions. One of a number of suitable filter design methods are used to compute the filter coefficient values that correspond to the desired spectral characteristic.

A Kaiser window filter design method is preferable for this application. Once the desired spectral shape is established, the filter coefficients are determined from the following equation:

\[ C_n = A_n \cos(2\pi f_n/k) W_n \]

In equation (24), \( C_n \) represents the \( n \)th filter coefficient, \( A_n \) represents samples of the desired spectral shape at frequencies \( f_n \), \( f_n \) represents the sampling frequency and \( W_n \) represents samples of the Kaiser Window. The spectral sample points, \( A_n \), are spaced at frequencies, \( f_n \), which are separated by the 6 dB bandwidth of the window, \( W_n \), so that a relatively smooth filter characteristic results that passes through each of the sample values. The frequency resolution and maximum slope of the frequency response of the resulting filter is determined by the number of coefficients or length of the filter. In the implementation shown in FIG. 8, filters F1 and F2 have a length of 30 taps which, at a sampling rate of 12.5 kHz, gives a frequency resolution of about 700 Hz and a maximum spectral slope of 0.04 dB/Hz.
Circuit 180 of FIG. 8 simplifies the fitting process. Through a suitable interactive display on a host computer (not shown), each spectral sample value $A_k$ is independently selected. While wearing a hearing aid which includes circuit 180 in a sound field, such as speech weighted noise at a given level, the patient adjusts each sample value $A_k$ to a preferred setting for listening. The patient also adjusts filter F2 to a preferred shape that is comfortable only for loud sounds.

Appendix A contains a program written for a Macintosh host computer for setting channel gain and limit values in a four channel contiguous band hearing aid. The filter coefficients for the bands are read from a file stored on the disk in the Macintosh computer. An interactive graphics display is used to adjust the filter and gain values.

In view of the above, it will be seen that the several objects of the invention are achieved and other advantageous results attained.

As various changes could be made in the above constructions without departing from the scope of the invention, it is intended that all matter contained in the above description or shown in the accompanying drawings shall be interpreted as illustrative and not in a limiting sense.
APPENDIX A

Program WDHA

Wearable Digital Hearing Aid Control Program V. 1.0
Central Institute For The Deaf
818 South Euclid Ave.
St. Louis Mo. 63110
Phone: 314-652-3200

Supported in part by:
The Rehabilitation Research And Development Service
Dept. of Medicine and Surgery; Veterans Administration

General Overview

A program entitled "WDHA" has been written for the Macintosh personal computer. When a wearable digital hearing aid is attached to the Macintosh's SCSI bus peripheral interface, the user of the WDHA program can alter the operation of the hearing aid via an easy to use Macintosh style user interface.

Using the WDHA Program

Starting The Program

Upon starting the program, the Macintosh interrogates the hearing aid to determine which program it is running. If the hearing aid responds appropriately, a menu containing the options which apply to that particular program appears in the menu bar. If no response is received from the hearing aid, the menu entitled "WDHA Disconnected" appears in the menu bar, as follows:

Should this menu appear, this indicates that there is some problem with the hearing aid. The source of this problem could be that the hearing aid is truly disconnected, that it is simply turned off, or that the hearing aid battery is dead. Upon correcting the problem,
choose the "New WDHA Program" menu entry to activate the proper menu for the hearing aid.

The Aid Parameters Window

The four channel hearing aid programs have the titles Aid12 through Aid14. Choosing the "Aid Parameters" menu entry will cause the aid parameters window to be displayed, as follows:

<table>
<thead>
<tr>
<th>Channel</th>
<th>Gain</th>
<th>Limit SPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>26</td>
<td>105</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>106</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>115</td>
</tr>
</tbody>
</table>

The bar graph and chart depict the current settings of the gains and limits for each channel of the hearing aid. A gain or limit setting can be changed by dragging the appropriate bar up or down with the mouse. The selected bar will blink when it is activated, and can be moved until the mouse is released, at which point the hearing aid is updated with the new values.

The control buttons indicate whether the hearing aid is on or off (i.e. whether the hearing aid program is running), and whether the input or output attenuators are switched on or off. Any of these settings can be changed simply by clicking on the appropriate buttons.

Ear Module Calibration

The File menu has an option called "Calibrate Ear Module" which should be used whenever the program is started or an ear module is inserted (or re-inserted) in a patient's ear. Proper use of
this option insures that the gains actually generated by the hearing aid are as close to the gains indicated by the program as possible.

The lower right hand corner of the Aid Parameters window displays the results of the most recent ear module calibration, including the name of the calibration file and the four Hc values, where Hc is the difference between the real ear pressure measured in the ear canal and the standard pressure measured on a Zwislocki at the center frequency of each channel. After choosing this option the user must open the file containing the ear module coefficients, by double clicking on the file's name, via a standard Macintosh dialog box:

The program will then play a series of four tones in the patient's ear, using the power measurement to determine the real pressure in the ear canal.

The file containing the ear module coefficients should be created with a text editor and saved as a text-only file. The file contains all the H values for a given ear module, separated by tabs, spaces, or carriage returns. It should begin with the four Hc values, followed by the Hr values, then Hc, and then Hp. The values entered for the Hc values can be arbitrary, since the program calculates them and stores them into the file. An ear module file as you would enter it might look as follows:

```
-100 -85 -90 -84 121 116 127 120
0
0
```
Here the first row contains both the four $He$ values and the four $Hr$ values. Following this are four zeros (since the $Hc$ values are unknown). The sixth row contains the $Hp$ values. Note that values are arbitrarily separated by tabs, spaces, or carriage returns.

After doing an ear module calibration with the program, the new $Hc$ values are displayed in the Aid Settings window, and also written to the same file, with the data re-formatted into a separate row for each $H$ value, as follows:

```
-100  -85  -90  -84
121   116  127  120
-5     -4   -10  0
-124  -121  -134  -143
```

The Tone Parameters Window

The four channel programs also have the ability to play pure tones for audiometric purposes. The Tone Parameters window is available to activate these functions. Choosing the "Tone Parameters" menu entry will cause the Tone Parameters window to be displayed, as follows:

```
Tone Parameters

<table>
<thead>
<tr>
<th>Tone burst count?</th>
<th>3</th>
<th>X Hearing Aid On</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rise time sample count?</td>
<td>309</td>
<td>Input Attenuation</td>
</tr>
<tr>
<td>Signal on sample count?</td>
<td>2455</td>
<td></td>
</tr>
<tr>
<td>Fall time sample count?</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>Signal off sample count?</td>
<td>3959</td>
<td></td>
</tr>
<tr>
<td>Frequency?</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>Atten re max out (dB)?</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Power = -12.816046</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Field Mike

Probe Mike

Start
```
Loading Filter Taps

The programs titled Aid13 and Aid14 have the capability to download filter tap coefficients to the hearing aid. The coefficients are read into memory from a text file which the user creates with any standard text editor. The coefficients in these files are signed integers such as "797" or "-174" (optionally be followed by a divisor, such as in "-12028/2") and must be seperated by spaces, tabs, or carriage returns.

The Aid13 program has 32 taps per filter, and the Aid14 program has 31 taps per filter, but since the filters are symmetric about the center tap you only provide half this number of taps, or 16 taps per filter. Thus the files contain 64 coefficients for the 4 channels. For example, the file titled TapsFour has the following format:

```
-535/4 -431/4 -254/4 0 333/4 743/4 1220/4 1750/4
-2315/4 2892/4 3545/4 3977/4 4432/4 4797/4 5052/4 5183/4
-34/2 -231/2 -223/2 0 292/2 398/2 777/2 -745/2
-1873/2 -2869/2 -3212/2 -2535/2 -831/2 1483/2 3683/2 5021/2
-83/2 502/2 859/2 0 -1128/2 -866/2 189/2 128/2
-442/2 890/2 3076/2 1605/2 -3814/2 -6280/2 -922/2 6543/2
```
The option to download coefficients is enabled by choosing the "Tap Filter Load" menu entries. The Macintosh will then present the standard open file dialog box, which you use to specify the name of the appropriate text file.

Program Design

The program is written in 68000 Assembly Language using the Macintosh Development System assembler, from Apple.

The program has been structured into separate managers for each of the program's functions. A separate file contains the functions associated with each manager. For example, the Parameter Settings (or "PS") manager is contained in the file WDHAPS.Asm, and includes all routines associated with the Aid Parameters window.

Below is a description of each manager, its function, and the routines contained in each.

WDHA.Asm

The overall program structure is typical of a Macintosh application in that it has an event loop which dequeues events from the event queue, and then branches to code which processes each particular type of event. WDHA.Asm contains the WDHA program's event loop.

WDHAPS.Asm

The Parameter Settings ("PS") manager contains all routines associated with the Aid Parameters window, which allows the user to control the gains and limits of each of the channels in the four channel programs. Specifically, these routines are as follows:

WDHAPSOpen - Create and display the Aid Parameters window.
WDHAPSClose - Close the Aid Parameters window and dispose the memory associated with it.
WDHAPSShow - Make the Aid Parameters window visible.
WDHAPSHide - Make the Aid Parameters window invisible.
WDHAPSDraw - Update the contents of the Aid Parameters window.
WDHAPSCControl - Cause the appropriate modification of the Aid Parameters window when a mousedown event occurs within it's content region.

WDHAPSSIS - Given a window pointer, this routine determines if it is the Aid Parameters window or not.

WDHAPSSetParam - Update the hearing aid to contain the settings specified in the Aid Parameters window.

WDHATCAsm

The TC manager contains all routines associated with the Tone Parameters window, which allows the user to specify the parameters for the test/calibrate function of the four channel program, and initiate the test. Specifically, these routines are as follows:

WDHATCOpen - Create and display the Tone Parameters window.

WDHATCClose - Close the Tone Parameters window and dispose the memory associated with it.

WDHATCShow - Make the Tone Parameters window visible.

WDHATCHide - Make the Tone Parameters window invisible.

WDHATCDraw - Update the contents of the Tone Parameters window.

WDHATCControl - Cause the appropriate modification of the Tone Parameters window when a mousedown event occurs within it's content region.

WDHATCIS - Given a window pointer, this routine determines if it is the Tone Parameters window or not.

WDHATCIdle - Blink the text caret of the Tone Parameters window.

WDHATCKey - Insert a key press into the active text box of the Tone Parameters window.

WDHATCDoTest - Initiate a test by the hearing aid program, using the parameters specified by the Tone Parameters window.

EarModuleCalibrate - Compute the Hc values for each of the four channels (this routine uses the test/calibrate function of the hearing aid to figure the real ear pressure at the center frequency of each channel).

WDHASCSI.Asm

The SCSI manager contains all routines which send record structures to the hearing aid via the SCSI bus.
SetParam - Send the four channel parameter record (containing the gains and limits) to the four channel hearing aid program.

SetCoefficients - Send out the filter tap coefficients to the four channel hearing aid program.

SetFileParams - Send the parameters required by the spectral shaping program.

wdhatest - Initiate a pure tone test by sending the test/calibrate record to the hearing aid.

WDHAFC.Asm

The WDHA program accesses some numerical values it needs by reading them in from text files. The File Coefficients (FC) manager contains routines which access these text files.

WDHAFCSet - This routine is called when the user selects the "Load Filter Taps" menu option. It uses the SFGetFile dialog to get the name of a text file containing filter coefficients, convert the contents to integer form, and then downloads them to the hearing aid.

WDHASetFileParams - This routine is used to download parameters to the Spectral Shaping hearing aid program. It uses the SFGetFile dialog to get the name of a text file containing the spectral shaping parameters, converts the contents to integer form, then downloads them to the hearing aid.

WDHACalEarModFile - This routine is called when the user calibrates the ear module. It uses the SFGetFile dialog to get the name of a text file containing ear module H Tables, and converts its contents to integer form in memory. Then it calibrates the ear module using the TC manager function EarModuleCalibrate. Finally, it writes the new H Tables over the same file.

WDHAMenu.Asm

The Menu manager contains all routines associated with the WDHA program's menu bar.

MakeMenus - Create the Menu bar containing the accessory, file, and hearing aid menus, and display it on the screen.
MenuBar - When the main event loop gets a mouseDown event located in the menu Bar, this routine calls the appropriate code to handle the selection.

SetProgMenu - This routine interrogates the hearing aid to determine which program it is currently running, then places the appropriate menu in the menu bar.

Programmer's Note -

As explained earlier, the WDHA program has separate pulldown menus defined for each program which runs on the hearing aid, giving the options available for that particular program. It is not difficult to add a new menu to the hearing aid program. The following example shows the steps one would follow to add a new aid menu (in this case 'Aid17') to the menu bar.

First of all, the constants needed for the menu must be defined with equate statements. You must define the code returned by the aid program when it is interrogated by the Macintosh, the identifier for the menu itself (as required by the NewMenu toolbox function), and the offset within the menu handles declarations where this handle will reside (the handles are defined in a sequential block of memory near the end of the Menu.Asm file).

Aid17ID equ -17 ; aid program id returned by interrogating the aid.
Aid17Menu equ 17 ; Unique menu identifier
menuaid17 equ 40 ; 10*4=menuhandle offset (this is the tenth handle)

Next you would declare the location to store the menu's handle at the end of the menu handles declarations:

dc.1 0 ; Aid17 menu handle

Next one would add code to the MakeMenus routine to create the new menu (simply cut and paste the code which creates one of the current menus and modify it accordingly).

You would also modify the SetProgMenu routine to handle the new menu (once again simply replicate the code sections which handle one of the old menus, and change the menu names appropriately).

Finally, you would modify the MenuBar routine to handle your new menu. If all the options contained in your menu are also in the
other hearing aid menus, you can call the InAidMenu procedure (as the other menus do), otherwise you must define your own procedure to call.

WDHADisk.Asm

The disk manager contains routines used to access disk files on the Macintosh.

DiskCreate - Create a new file.
DiskRead - Read sectors from a file.
DiskWrite - Write sectors to a file.
DiskEject - Eject a disk.
DiskOpen - Open a file.
DiskClose - Close a file
DiskSetFPos - Set the position of a file's read/write mark.
DiskSetEOF - Set the location of the end of file marker for a file.
DiskSetFInfo - Set the finder information for a file.
Include MacTraps.D
Include Too1EquX.D
Include SysEquXD.
Include QuickEquX.D
include MDS2:WDHAPS.hdr
Include MDS2:WDHATC.hdr
Include MDS2:WDHAMenu.hdr

WDHA program
This program controls several Macintosh windows which allow the user to manipulate the digital hearing aid. The Macintosh communicates with the aid by sending records via the SCSI port.

This particular file is a "standard" Macintosh style event loop which dequeues each event and calls the appropriate routine to handle the event.

Additional files contain routines associated with each control window.

Executing the program should provide an overall understanding of the function of these windows. Specifically, the packages used are:

The WDHA Parameter Settings Window Manager - in WDHAPS.Asm
The WDHA Test/Calibrate Window Manager - in WDHATC.Asm

In addition, the following files contain various utility routines:

WDHAMenu.Asm - sets up the menus
WDHASCASI.Asm - low level routines for communicating through the SCSI bus
WDHAFC.Asm - contains high-level routines for downloading coefficient files to the hearing aid.
WDHADisk.Asm - routines for doing disk access.

External Definitions

XDEF Start
XDEF EventLoop
XDEF Update
XDEF What
XDEF When
XDEF EventRecord
XDEF WWindow
XDEF Message
XDEF Where
XDEF Modify

Constant Definitions

ActiveBit equ 0 :Bit position of deactivate in modify

Code Starts Here

Start:
bsr InitManagers ; Initialize ToolBox
bsr WDHAPSOpen ; Create the parameter settings window.
bsr WDHATCOpen ; Create the test/calibrate window.
bsr WDHATCHide ; Don't leave it open though.
bsr MakeMenus ; Set up the menus

EventLoop:

 SystemTask ; Give System some time
bsr WHDATCIdle ; Blink the test window's caret

FUNCTION GetNextEvent(eventMask: INTEGER; VAR theEvent: EventRecord): BOOLEAN

CLR -(SP); Clear space for result
MOVE $0FFF,(SP); Allow 12 low events
PEA EventRecord; Place to return results
_GetNextEvent ; Look for an event
MOVE (SP)+,DO; Get result code
BEQ EventLoop; No event... Keep waiting
BSR HandleEvent; Go handle event
bra EventLoop; return to eventloop call

HandleEvent:
; Use the event number as an index into the Event table. These 12 events
; are all the things that could spontaneously happen while the program is
; in the main loop.

MOVE What,DO ; Get event number
ADD D0,D0 ; *2 for table index
MOVE EventTable(DO),D0 ; Point to routine offset
JMP EventTable; and jump to it

EventTable:

DC.W OtherEvent-EventTable ; Null Event (Not used)
DC.W MouseDown-EventTable ; Mouse Down
DC.W OtherEvent-EventTable ; Mouse Up (Not used)
DC.W KeyEvent-EventTable ; Key Down
DC.W OtherEvent-EventTable ; Key Up (Not used)
DC.W KeyEvent-EventTable ; Auto Key
DC.W Update-EventTable ; Update
DC.W OtherEvent-EventTable ; Disk (Not used)
DC.W Activate-EventTable ; Activate
DC.W OtherEvent-EventTable ; Abort (Not used)
DC.W OtherEvent-EventTable ; Network (Not used)
DC.W OtherEvent-EventTable ; VO Driver (Not used)

------------------------------- Event Actions ---------------------------

OtherEvent:

rts

Activate:
; An activate event is posted by the system when a window needs to be
; activated or deactivated. The information that indicates which window
; needs to be updated was returned by the NextEvent call.

bst #ActiveBit,Modify ; Activate?
beg Deactivate ; No, go do Deactivate
; Bring it to the front
move.l Message,(sp)
Update:
; The window needs to be redrawn.

PROCEDURE BeginUpdate (theWindow: WindowPtr);
    MOVE message,-(SP) ; Get pointer to window
    _BeginUpdate
    MOVE message,-(SP) ; Begin the update
    bsr WDHATCIS
    tst. w (sp)+
    BEQ DonTCDraw
    bsr WDHCATDraw
    bra DoneDraw
DonTCDraw:
    MOVE message,-(sp) ; Was it our TC window?
    bsr WDHAPIS
    tst. w (sp)+
    BEQ DonPSDraw
    bsr WDHAPSDraw
    bra DoneDraw
DonPSDraw:
    Procedure EndUpdate (theWindow: WindowPtr);
    MOVE message,-(SP) ; Get pointer to window
    _EndUpdate
    RTS

MouseDown:
; If the mouse button was pressed, we must determine where the click
; occurred before we can do anything. Call FindWindow to determine
; where the click was; dispatch the event according to the result.

FUNCTION FindWindow (thePt: Point; VAR whichWindow: WindowPtr; INTEGER);
    CLR -(SP) ; Space for result
    MOVEL Where,-(SP) ; Get mouse coordinates
    PEA WWindow ; Event Window
    _FindWindow
    MOVE (SP)+,D0 ; Get region number
    ADD D0,D0 ; *2 for index into table
    MOVE WindowTable(D0),D0 ; Point to routine offset
JMP WindowTable(D0) ; Jump to routine

WindowTable:
  DC.W other-WindowTable ; In Desk (Not used)
  DC.W MenuBar-WindowTable ; In Menu Bar
  DC.W SystemEvent-WindowTable ; System Window (Not used)
  DC.W Content-WindowTable ; In Content
  DC.W Drag-WindowTable ; In Drag
  DC.W Grow-WindowTable ; In Grow
  DC.W GoAway-WindowTable ; In Go Away

Other:
  rts

SystemEvent:
  ; Call SystemClick to handle the desk accessory windows.
  pea EventRecord
  move.l window, -(sp)
  SystemClick
  rts

Content:
  ; Was it in the content of an active window?
  clr.l -(sp)
    FrontWindow
    move.l (sp)+, d1 ; Get the FrontWindow in d1
    cmp.l wwindow, d1 ; Are they the same?
    beq WasActive
    move.l wwindow, -(sp) ; It wasn't
    SelectWindow
    bra DoneContent
  
  WasActive:
    move.l wwindow, -(sp)
    bsr WDAPSSIS ; Was it our PS window?
    tst.w (sp)+
    beq NotPSContent
    move.l where, -(sp) ; Handle the event.
    bra DoneContent

  NotPSContent:
    move.l wwindow, -(sp)
    bsr WDAPSSIS ; Was it our TC window?
    tst.w (sp)+
    beq NotTCContent
    move.l where, -(sp) ; Handle the event
    bsr WDAPSSControl
    bra DoneContent

  NotTCContent:
    DoneContent:
      rts

Drag:
  ; The click was in the drag bar of the window. Draggit.
  DragWindow ((theWindow:WindowPtr; startPt: Point; boundsRec: Rect);
47

MOVEF wwindow, -(SP); Pass window pointer
MOVEF where, -(SP); mouse coordinates
PEA bound ;and boundaries
_DragWindow ; Drag Window
rts

Grow:
; The click was in the grow box
NoGrow: rts

GoAway:
cir.b -(sp)
move.f wwindow, -(sp)
move.f where, -(sp)
_TrackGoAway
lst.b (sp)+
boq NoGoAway
; Close the Window
; make room for a Boolean
; Track It
; Did they stay in the box?
; If no then don't close.

JustHide:
; PROCEDURE HideWindow (theWindow: WindowPtr)
MOVEF wwindow, -(SP)
_HideWindow
rts

NoGoAway:

keyEvent:
; Space for result
; Get window pointer on stack
bsr WDHATCIS
lst.w (sp)+
boq TCNotActive
move.w message+2, -(sp)
bsr WDHATCKey
; get the char
; Insert it in the active text box
TCNotActive:

; InitManagers initializes all the ToolBox managers. You should call
; InitManagers once at the beginning of your program if you are using
; any of the ToolBox routines.
InitManagers:
pea -4(a5)
_InitGraf
_InitFonts
move.f $0000FFFF, d0
_FlushEvents
_InitWindows
_InitMenus
cir.i -(sp)
_InitDailogts
_TEdinit
_InitCursor
rts
; WDHA header file
; this file must be included to access the data structures contained in
; the file WDHA.Asm
XREF EventLoop
XREF Update
XREF EventRecord
XREF What
XREF Message
XREF When
XREF Where
XREF Modify
XREF WWindow

TRUE EQU 1
FALSE EQU 0
## Dialog

**Macro**

```assembly
Macro Dialog xpos,ypos,txtstring,result =
  move.w[xpos].-(SP)
  move.w[ypos].-(SP)
  _MoveTo
  pea '[txtstring]'
  _DrawString
  pea KeyBuf
  bsr GetStr

lea keybuf.a0
move.w#1.-{SP}
    ;StringToNum
move.wd0.(result)
```

## DispString

**Macro**

```assembly
Macro DispString xpos,ypos,txtstring =
  move.w[xpos].-(SP)
  move.w[ypos].-(SP)
  _MoveTo
  pea '[txtstring]'
  _DrawString
```

## DispValue

**Macro**

```assembly
Macro DispValue xpos,ypos,label,value =
  movem.l a0-a6/d0-d7,-{sp}
  move.w[xpos].-(SP)
  move.w[ypos].-(SP)
  _MoveTo
  pea '[label]'
  _DrawString

lea KeyBuf.a0
move.l (value),d0
move.w#0.-{SP}
    ;Select NumToString
    ;Pack7

pea KeyBuf
  _DrawString
movem.l (sp)+,a0-a6/d0-d7
```

## DispWValue

**Macro**
Macro DispWValue xpos,ypos,label,value =
movem.l a0-a6/d0-d7-(sp)
mov.e w{xpos},-(SP)
mov.e w{ypos},-(SP)
_MoveTo
lea '(label)'
_DrawString
les. KeyBuf,a0
move.w{value},d0
eql. d0
move.w#0,-(SP) ;Select NumToString
_Pack7

lea KeyBuf
_DrawString
movem.l (sp)+,a0-a5/d0-d7
; WDHAMenu.Asm
; This file contains routines which create and manipulate the menus used in
; the WOHA program.

Include MacTraps.D
Include ToolEqxD.D
Include SysEqx.D
Include QuickEqxD.D
Include MDS2:WDHAMac.txt
Include MDS2:WDHA.hdr
Include MDS2:WDHAPIS.hdr
Include MDS2:WDHAFC.hdr
Include MDS2:WDHASCSI.hdr

xdef MakeMenus
xdef MenuHandles
xdef MenuBar

AppleMenu EQU 1
AboutItem EQU 1
menuapple equ 0 ; menuhandle offset

QuitItem EQU 2
menufile equ 4 ; menuhandle offset

; Now the aid menus. All have a 'new program' entry, and a blank line.
NewProgItem EQU 1
AidBlank EQU 2

Aid12ID EQU -12 ; program version id
Aid12Item EQU 5
SetItem EQU 3
TestItem EQU 4
menuaid12 equ 8 ; menuhandle offset

Aid13ID EQU -13 ; program version id
Aid13Item EQU 6
FCItem EQU 5
menuaid13 equ 12 ; menuhandle offset

Aid14ID EQU -14 ; program version id
Aid14Item EQU 7
menuaid14 equ 16 ; menuhandle offset

SS15ID EQU -100
SS15Menu EQU 8
LoadItem EQU 3
menuss15 equ 20

NoneMenu EQU 9
menunone equ 24
; Name: MakeMenus
; Function: MakeMenus creates and displays the menu bar.
; Input: None
; Output: None

MakeMenus:
    ; Clear menu bar
    _ClearMenuBar

lea MenuHandles, a4
; First add Apple Menu
; Make it:
    clr.1 -(sp)
    move.w #AppleMenu,-(sp) ; first menu
    pea AppleName ; apple character
    NewMenu
    move.1 (sp)+, menuapple(a4) ; store handle
    ; Add entries
    move.1 menuapple(a4),-(sp) ; push handle again
    pea 'About WDHA:;' ; push menu item
    _AppendMenu
    move.1 menuapple(a4),-(sp) ; push handle again
    move.1 #DRVR-,-(sp) ; load all drivers
    _AddResMenu
    ; Insert it in the menu bar.
    move.1 menuapple(a4),-(sp) ; push handle again
    move.w #0, -(sp) ; insert at end
    _InsertMenu

; Now add File Menu
; Make it:
    clr.1 -(sp)
    move.w #FileMenu,-(sp) ; second menu
    pea 'File' ; menu title
    NewMenu
    move.1 (sp)+, menufile(a4) ; store handle
    ; Add entries
    move.1 menufile(a4),-(sp) ; push handle again
    pea 'Quit' ; push menu item
    _AppendMenu
    ; Insert it in the menu bar.
    move.1 menufile(a4),-(sp) ; push handle again
    move.w #0, -(sp) ; insert at end
    _InsertMenu

; Now create the WDHA program menus.
; none
    clr.1 -(sp)
    move.w #NoneMenu,-(sp)
    pea 'WDHA Disconnected' ; menu title
    NewMenu
    move.1 (sp)+, menunone(a4) ; store handle
    ; Add entries
    move.1 menunone(a4),-(sp) ; push handle
    pea New WDHA Program: ; menu items
AppendMenu

: aid12

clr.1 -(sp) ;space for function result
move.w#Aid12Menu,-(sp) ;menu title
pea 'Aid12' ;_NewMenu
move.(sp)+,menusaid12(a4) ;store handle

;Add entries.
move.1 menusaid12(a4),-(sp) ;push handle
pea 'New WDHA Program;(-4 Channel Parameters;Test Calibrate':menu items.

AppendMenu

: aid13

clr.1 -(sp) ;space for function result
move.w#Aid13Menu,-(sp) ;menu title
pea 'Aid13' ;_NewMenu
move.(sp)+,menusaid13(a4) ;store handle

;Add entries.
move.1 menusaid13(a4),-(sp) ;push handle
pea 'New WDHA Program;(-4 Channel Parameters;Test Calibrate;32 Tap Filter Load':menu items.

AppendMenu

: aid14

clr.1 -(sp) ;space for function result
move.w#Aid14Menu,-(sp) ;menu title
pea 'Aid14' ;_NewMenu
move.(sp)+,menusaid14(a4) ;store handle

;Add entries.
move.1 menusaid14(a4),-(sp) ;push handle
pea 'New WDHA Program;(-4 Channel Parameters;Test Calibrate;31 Tap Filter Load':menu items.

AppendMenu

: SS15

clr.1 -(sp) ;space for function result
move.w#SS15Menu,-(sp) ;menu title
pea 'SS15' ;_NewMenu
move.(sp)+,menus15(a4) ;store handle

;Add entries.
move.1 menus15(a4),-(sp) ;push handle
pea 'New WDHA Program;(-14 Parameter Load':menu items.

AppendMenu

; Insert one in the menu bar since SetProgMenu deletes one.
move.1 menu1one(a4),-(sp) ;push handle again
move.w#0,-(sp) ;insert at end

_InserMenu

; Set the proper WDHA program menu
bsr SetProgMenu
rts

; Name: SetProgMenu
; Function: This routine interrogates the hearing aid to determine which
; program it is currently running, then places the appropriate menu
; in the menu bar.
; Input: None
; Output: None
SetProgMenu:
  Close windows so that no inappropriate windows remain.
  bar WDPHAPHide
  bsr WDPHATCHide
; Delete the old menu (whichever it is)
  move.w #Aid12Menu,-(sp)
  _DeleteMenu
  move.w #Aid13Menu,-(sp)
  _DeleteMenu
  move.w #Aid14Menu,-(sp)
  _DeleteMenu
  move.w #SS15Menu,-(sp)
  _DeleteMenu
  move.w #NoneMenu,-(sp)
  _DeleteMenu
; Default to NoneMenu
  lea MenuHandles.a4
  move.w menuone(a4),-(sp)
  move.w #0,-(sp)
  _InsertMenu
; redraw the bar
  _DrawMenuBar
  move.w #0,-(sp) ; clear any highlighting.
  _HILiteMenu
; Now check what it is
  clr.w -(sp)
  bsr SC5Interrogate
  move.w(sp)+, d0
  lea MenuHandles.a4
  cmp.w #Aid12ID,d0
  bne NotAid12
  move.w menuaid12(a4),a3 ; get handle
  bra AddProgMenu
 NotAid12:
  cmp.w #Aid13ID,d0
  bne NotAid13
  move.w menuaid13(a4),a3 ; get handle
  bra AddProgMenu
 NotAid13:
  cmp.w #Aid14ID,d0
  bne NotAid14
  move.w menuaid14(a4),a3 ; get handle
  bra AddProgMenu
 NotAid14:
  cmp.w #SS15ID,d0
  ;
bne NotSS15
move.l menusu15(a4),a3 ;get handle
bra AddProgMenu

NotSS15:
move.l menuu0n0ne(a4),a3
move.w #20,(sp)
_SysBeep

AddProgMenu:
move.w #NoneMenu,(sp)
_DeleteMenu
move.l a3,-(sp)
move.w #00,(sp)
_InsertMenu
: redraw the bar
_DrawMenuBar

ClearReturn:
move.w #00,(sp) ;clear any highlighting.
_HLilteMenu

rts

; Name: MenuBar
; Function: This routine should be called when the mouse is clicked in the
; menu bar.
; Input: None
; Output: None

MenuBar:
clr.l -(sp) ;space for result
move.l where,-(sp) ;location of mouse
_MenuSelect
move.l (sp)+,d0 ;get result (menu id, item #)
swap d0 ;get menu id in low word

Choices:
cmp.w #0,d0 ;Was it in any menu?
beq @1 no menu id
cmp.w #AppleMenu,d0 ;Was it in the apple menu?
beq InAppleMenu
.cmp.w #FileMenu,d0 ;Was it in the file menu?
beq InFileMenu
cmp.w #NoneMenu,d0
beq InNoneMenu
cmp.w #AldMenu,d0
beq InAldMenu
cmp.w #Ald1Menu,d0
beq InAld1Menu
cmp.w #Ald12Menu,d0
beq InAld12Menu
cmp.w #Ald13Menu,d0
beq InAld13Menu
cmp.w #Ald14Menu,d0
beq InAld14Menu
cmp.w #SS15Menu,d0
beq InSS15Menu
cmp.w #SSMenu,d0
beq InSSMenu

@1 bra ClearReturn

InAppleMenu:
; Get item
swap d0 ; get item # in low word

cmp.w #AboutItem.d0
jne NotAbout!

Open About dialog window.

:FUNCTION NewWindow (wStorage: Ptr; boundsRect: Rect;
  title: Str255; visible: BOOLEAN;
  procID: INTEGER; behind: WindowPtr;
  goAwayFlag: BOOLEAN;
  refCon: LongInt) : WindowPtr;

  SUBQ $4,SP ; Space for function result
  CLLRL (-SP) ; Storage for window (Heap)
  PEA 'AboutWDHA' ; Window title
  PEA AboutBounds ; Window position
  MOVEM #-255,-(SP) ; Make window visible
  MOVE #dBoxProc,-(SP) ; Standard document window
  MOVEM #1,-(SP) ; Make it the front window
  move.B #1,-(SP) ; Window has goAway button
  CLLRL - (SP) ; Window refCon
  _NewWindow lea AboutPrt,a4
  MOVE (SP)+,(a4) ; Save handle for later
  MOVE (a4),-(SP) ; Make sure the new window is the port

:PROCEDURE SetPort (gp: GrafPort)
  _SetPort
  move.w #0,-(sp) ; Make sure it's the system font
  move.w#1,-(sp) ; Bold
  _TextFace
  DispString #20,#16,Wearable Digital Hearing Aid Fitting Procedure V. 1.0
  move.w#0,-(sp) ; Plain Text
  _TextFace
  DispString #200,#32,Central Institute For The Deaf
  DispString #200,#48,818 South Euclid Ave.
  DispString #200,#64,St. Louis Mo. 63110
  DispString #200,#80,Phone: 314-652-3200
  move.w#1,-(sp) ; Bold
  _TextFace
  DispString #200,#96,Supported in part by:
  move.w#0,-(sp) ; Plain Text
  _TextFace
  DispString #40,#112,The Rehabilitation Research And Development Service
  DispString #40,#128,Dept. of Medicine and Surgery: Veterans Administration

; Print the big "CID"
  move.w#36,-(sp)
  _TextSize
  move.w#17,-(sp) ; Bold+Shadow
  _TextFace
  DispString #44,#64,CID

; Set text characteristics back to normal
  move.w#12,-(sp)
  _TextSize
  move.w#0,-(sp) ; Plain Text
  _TextFace

; Wait for an event
move.l #$0000FFFF,d0
_FlushEvents

EvtWait:

; FUNCTION GetNextEvent(eventMask: INTEGER:)
VAR theEvent: EventRecord: BOOLEAN
CLR (SP): Clear space for result
MOVE #$00FF, (SP): Allow 12 low events
PEA EventRecord: Place to return results
_GetNextEvent
MOVE (SP)+,d0: Get result code
BDW EvtWait: No event... Keep waiting

; Dispose Window
move.l AboutPtr,-(sp)
_DisposeWindow
bra ClearReturn

NotAbout:
lea MenuHandles,a4
move.l menuapple(a4),- (sp): Look in Apple Menu
move.w d0,-(sp): what item #
pea DeskName: get item name
_GetItem
clr.w -(sp): space for result
pea DeskName: open DeskName acc
_OpenDeskAcc
move.w (sp)+,d0: pop result
bra ClearReturn

InFileMenu:
swap d0: get item # in low word
cmp.w #QuitItem,d0: Is it quit?
bsr DoneFile: if not forget it
bsr WDHATCClose: dispose of the test/calibrate window
 bsr WExitToShell: leave application

DoneFile:
bra ClearReturn

InAidMenu:
swap d0: get item # in low word
cmp.w #NewProgltem,d0
bne @9
bsr SetProgMenu
bsr WMDone

@9
cmp.w #SetItem,d0
bne @1
bsr WDHAPSShow
bsr WMDone

@1
cmp.w #TestItem,d0
bne @2
bsr WDHATCShow
bsr WMDone

@2
cmp.w #FCItem,d0
bne @4
bsr WOHAFCSet
bra WMDone
@4
WMDone bra ClearReturn

InSSMenu:
    swap d0 ; get item # in low word
    cmp.w #NewProgItem,d0
    bne @1
    bsr SetProgMenu
    bra SSDone
@1
    cmp.w #LoadItem,d0
    bne @2
    bsr WOHASetFileParams
    bra SSDone
@2
SSDone bra ClearReturn

------------------------ Data starts here ------------------------

MenuHandles:
    dc.l 0 ; handle to apple menu
    dc.l 0 ; handle to file menu
    dc.l 0 ; handle to aid12 menu
    dc.l 0 ; handle to aid11 menu
    dc.l 0 ; handle to aid14 menu
    dc.l 0 ; handle to aid15 menu
    dc.l 0 ; handle to none menu

AppleName: dc.b 1,314 ; A string containing the apple symbol
DeskName:   dc.b.w 16,0 ; desk accessories name
AboutPtr   dc.l 0 ; the About dialog window pointer
AboutBounds:
    dc.w 100 ; upper
    dc.w 50 ; left
    dc.w 232 ; lower
    dc.w 472 ; right
WDMenu header file

This file must be included if any routines in WDMenu are used.

\texttt{xref MakeMenus}
\texttt{xref MenuHandles}
\texttt{xref MenuBar}
; file WDHAPS_Asm

 Include MacTraps.D
 Include ToolEquX.D
 Include SysEqX.D
 Include QuickEqX.D
 Include SANEMacs.txt
 Include MDS2:WDHA.hdr
 Include MDS2:WDHASCS1.hdr

; WDHAPSParameter Settings Window Manager
; This package contains routines to manipulate the WDHAP Parameter
; Settings window. This window contains an interface which controls the
; gain and limit of each channel of the WDHA by allowing the user to move
; bars on a graph of Frequency versus dB SPL (execute the program for a better
; understanding), this control is referred to as the "PSGraph" in the program
; documentation. Next to this graph is a chart (the "PSChart") containing the
; numeric values of each channel's gain and limit.
; It also contains control buttons to specify if the WDHA should be in
; Hearing aid mode, if the input attenuation should be off or on, and whether
; the aid should use the probe mike or the field mike. The output attenuation
; is automatically turned on or off by the program, its control being used
; as an indicator of this status.
; Wherever the documentation refers to the term "theta", it is referring
; to the height of the lower bar of the bar graph, and wherever the documentation
; uses "phi", it refers to the height of the upper bar.

; External Definitions

XDEF WDHAPSSOpen
XDEF WDHAPSSClose
XDEF WDHAPSShow
XDEF WDHAPSHide
XDEF WDHAPSSDraw
XDEF WDHAPSCntrol
XDEF WDHAPSBS
XDEF WDHAPSSetParam

; Constant Definitions

CHANNELS EQU 4 ; There are four channels

PSG = The Parameter Settings Graph
PSGHeight EQU 120 ; Graph height in pixels
PSGChanWidth EQU 20 ; each bar is PSGChanWidth pixels wide.
PSGWidth EQU CHANNELS*PSGChanWidth ; Graph width in pixels
PSGinitX EQU 30 ; initial X coord (local) of ul corner of graph
PSGinitY EQU 20 ; initial Y coord (local) of ul corner of graph

PSC = The Parameter Settings Chart
PSCFHeight EQU PSGHeight/(CHANNELS+1) ; height of box in chart
PSCFWidth EQU 3*PSCFWidth
PSCinitX EQU PSGinitX+PSGWidth ; X coord (local) of ul corner of chart
PSClnitX EQU PSGlnitX ; Y coord (local) of ul corner of chart

; PS = The Parameter Settings Window
PSlnitX EQU 60 ; initial X coord (global) of upper left corner
PSlnitY EQU 80 ; initial Y coord (global) of upper left corner
PSRight EQU PSlnitX+PSGWidth+PSCWidth+2*PSGlnitX+140
PSBtnsSize EQU 12

; PSC = The Control Buttons
PSClnitX EQU PSGlnitX+PSGWidth+PSCWidth+10
PSClnitY EQU PSGlnitY+5
PSCbtnsHeight EQU PSCHeight

;------------------------- Subroutine Declarations -------------------------
; Name: WDHAPSOpen
; Function: Call this routine to create and display the PS Window.
; Input: None
; Output: None
WDHAPSopen:
        movem 1 d0-d2/a0-a6,-(sp) ; save registers
        Subrument Declarations

; FUNCTION NewWindow (wStorage: Ptr: boundsRect: Rect;
;                     title: Str255; visible: BOOLEAN;
;                     procID: INTEGER; behind: WindowPtr;
;                     goAwayFlag: BOOLEAN;
;                     relCon: LongInt) : WindowPtr;
; SUBR #4,SP       ; Space for function result
; CURL -(SP)      ; Storage for window (Heap)
; PEA WDHAPSBounds ; Window position
PEA 'WDHA Parameter Settings' ; Window title
MOVE.B #255,-(SP) ; Make window visible
MOVE #DocProc, -(SP) ; Standard document window
MOVEL #1,-(SP)    ; Make it the front window
move.B #1,-(SP)  ; Window has goAway button
CURL -(SP)       ; Window relCon
NEWWindow 
lea WDHAPSPtr,a4 
MOVEL (SP)+,(a4) ; Save handle for later
MOVEL (a4),-(SP) ; Make sure the new window is the port
; Procedure SetPort (gp: GrafPort)
; SetPort ; Make it the current port

; Name: WDHAPSClose
; Function: Call this routine to destroy the PS Window and remove it from
; the screen.
; Input: None
; Output: None
WDHAPSclose:
        movem l d0-d7/a0-a6,-(sp) ; save registers
move.l WDHAPSPtr,-(sp)
  KillControls

; Dispose Window
move.l WDHAPSPtr,-(sp)
  DisposeWindow
movem.l (sp)+,d0-d7/a0-a6 ; restore registers
  rts

; Name: WDHAPSShow
; Function: This routine makes the PS window visible and frontmost.
; Input: None
; Output: None

WDHAPSShow:
  movem.l d0-d7/a0-a6,-(sp) ; save registers
  Bring it to the front
  movem.l WDHAPSPtr,-(sp)
  BringToFront
  Show Window
  movem.l WDHAPSPtr,-(sp)
  ShowWindow
  movem.l WDHAPSPtr,-(sp)
  SelectWindow ; So select it.
  movem.l (sp)+,d0-d7/a0-a6 ; restore registers
  rts

; Name: WDHAPSHide
; Function: This routine makes the PS window invisible, removing it from the
; screen (but not destroying it).
; Input: None
; Output: None

WDHAPSHide:
  movem.l d0-d7/a0-a6,-(sp) ; save registers
  Hide Window
  movem.l WDHAPSPtr,-(sp)
  HideWindow
  movem.l (sp)+,d0-d7/a0-a6 ; restore registers
  rts

; Name: WDHAPSDraw
; Function: This routine draws the PS window's contents.
; Input: None
; Output: None

WDHAPSDraw:
  movem.l d0-d7/a0-a6,-(sp) ; save registers
  lea WDHAPSPtr.a4 ; Pointer on stack
  MOVEU (a4),-(sp)
  PROCEDURE SetPort (gp: GrafPort)
    SetPort ; Make it the current port
    First draw the graph
    push WDHAPSGraph
    _EraseRect ; clear it
    push WDHAPSGraph
    _FrameRect
    move.w #patTOr,-(sp)
PenMode
    move.w#0,d4
    ; change to Or pen mode.
DrawChans:
    cmp.w #CHANNELS,d4
    ; count thru channels
    beq DoneDC
    ; draw each channel
    ; done yet?
    move.wd4,-(sp)
    bsr CalThetaRect
    ; Calculate theta rectangle
    pea ThelaPat
    : Draw Theta Bar
    pea ThetaPat
    _PenPat
    move.wd4,-(sp)
    bar CalThetaRect
    pea TRect
    _PaintRect
    ; Fill with pattern
    add.w #1,d4
    bra DrawChans
DoneDC:
    _PenMode
    ; Reset Pen to original settings
    move.w#PSTextSize,-(sp)
    _TextSize
    move.w#PSGInitX+0*PSGChanWidth+PSGChanWidthV2,-(sp)
    move.w#PSGInitY+PSGHeight+PSTextSize,-(sp)
    _MoveTo
    move.w#1',.(sp)
    _DrawChar
    move.w#PSGInitX+1*PSGChanWidth+PSGChanWidthV2,-(sp)
    move.w#PSGInitY+PSGHeight+PSTextSize,-(sp)
    _MoveTo
    move.w#2',.(sp)
    _DrawChar
    move.w#PSGInitX+2*PSGChanWidth+PSGChanWidthV2,-(sp)
    move.w#PSGInitY+PSGHeight+PSTextSize,-(sp)
    _MoveTo
    move.w#3',.(sp)
    _DrawChar
    move.w#PSGInitX+3*PSGChanWidth+PSGChanWidthV2,-(sp)
    move.w#PSGInitY+PSGHeight+PSTextSize,-(sp)
    _MoveTo
    move.w#4',.(sp)
    _DrawChar
    move.w#PSGInitX+CHANNELS/2*PSGChanWidth+PSGChanWidthV2,-(sp)
    move.w#PSGInitY+PSGHeight+2*PSTextSize,-(sp)
    _MoveTo
    pea "Channel"
    _DrawString
    move.w#PSGInitX-20,-(sp)
    move.w#PSGInitY+PSGHeightV2-PSTextSize,-(sp)
    _MoveTo
pea 'dB'
_DrawString
movew.p$SInitX-24,-(sp)
movew.p$SInitY+$SHeight/2,-(sp)
_MoveTo
pea 'SPL'
_DrawString
movew.8,-(sp)
_TextSize
movew.p$SInitX-9,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.8'0',-(sp)
_DrawChar
movew.p$SInitX-20,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
pea '120'
_DrawString
; Now draw the chart.
_PenNormal
pea W$CHAPSCChart
_FrameRect
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_LineTo
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_LineTo
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_LineTo
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_LineTo
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_LineTo
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_LineTo
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_LineTo
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_LineTo
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_LineTo
movew.p$SInitX,-(sp)
movew.p$SInitY+$SHeight,-(sp)
_MoveTo
movew.p$SInitX+$SWidth,-(sp)
movew.p$SInitY+$SHeight,-(sp)

5,706,352
move.w #PSCInitY, -(sp)
_MoveTo
move.w #PSCInitX+2*PSCFWidth, -(sp)
move.w #PSCInitY+PSGHeight, -(sp)
_LineTo
move.w #PSCInitX+6, -(sp)
move.w #PSCInitY+PSCFHeight-6, -(sp)
_MoveTo
lea 'Channel'
_DrawString
move.w #PSCInitX+PSCFWidth+11, -(sp)
move.w #PSCInitY+PSCFHeight-6, -(sp)
_MoveTo
lea 'Gain'
_DrawString
move.w #PSCInitX+2*PSCFWidth+10, -(sp)
move.w #PSCInitY+PSCFHeight-6, -(sp)
_MoveTo
lea 'Limit'
_DrawString
move.w #CHANNELS.d4 ; Now draw the chart data with PrintVal
lea Theta3.a0 ; will draw the gains and limits too

DrChartNums:
; Draw channel #
move.w #0, -(sp) ; Column 0
move.w #d4, -(sp) ; Row is same as channel
.move.w (#aO),-(sp) ; value is channel

; Draw gain
move.w #1, -(sp) ; now do gain
move.w #d4, -(sp) ; Row is same as channel
move.w (#aO),-(sp) ; Show the theta value as gain

; Draw limit
move.w #2, -(sp) ; now do limit
move.w #d4, -(sp) ; Row is same as channel
move.w (#aO),-(sp) ; Show the Phi value as limit

; Draw the control buttons.
lea WDHA.PSPtr, -(sp) ; the window ptr
.DrawControls
ber WDHA.PSPtrParam ; update the WDHA.
move.l (sp)+, d0-d7/a0-a8 ; restore registers

; Name: PSAddControls
; Function: This routine adds the PS window's controls.
; Input: None
; Output: None
PSAddControls:
movem.l d0-d7/a0-a6, -(sp) ; save registers
; Set up the controls bounding rectangle.
lea TRect,a4
move.w#PSCInitY+0*PSCHeight,(a4) ; store y coord
move.w#PSCInitX,2(a4) ; store x coord
move.w#PSCInitY+2*PSCHeight+20,4(a4) ; store y coord
move.w#PSRight,6(a4) ; store x coord

; Push parameters for NewControl
clri .(a4) ; NewControl returns a handle
move.l WDHAPSPtr,-(sp) ; the window ptr
lea TRect ; the rectangle bounding the control
lea 'Hearing Aid On' ; title
move.b #TRUE,-(sp) ; visible
move.w#0,-(sp) ; value
move.w#0,-(sp) ; min
move.w#1,-(sp) ; max
move.w#1,-(sp) ; check box proc id
move.l #0,-(sp) ; refcon not used

; Call NewControl
_Call NewControl
lea AidControl,a3
move.l (sp)+,(a3) ; store the result

; Set up the controls bounding rectangle.
lea TRect,a4
move.w#PSCInitY+1*PSCHeight,(a4) ; store y coord
move.w#PSCInitX,2(a4) ; store x coord
move.w#PSCInitY+1*PSCHeight+20,4(a4) ; store y coord
move.w#PSRight,6(a4) ; store x coord

; Push parameters for NewControl
clri .(sp) ; NewControl returns a handle
move.l WDHAPSPtr,-(sp) ; the window ptr
lea TRect ; the rectangle bounding the control
lea 'Input Attenuation' ; title
move.b #TRUE,-(sp) ; visible
move.w#0,-(sp) ; value
move.w#0,-(sp) ; min
move.w#1,-(sp) ; max
move.w#1,-(sp) ; check box proc id
move.l #0,-(sp) ; refcon not used

; Call NewControl
_Call NewControl
lea IAControl,a3
move.l (sp)+,(a3) ; store the result

; Set up the controls bounding rectangle.
lea TRect,a4
move.w#PSCInitY+2*PSCHeight,(a4) ; store y coord
move.w#PSCInitX,2(a4) ; store x coord
move.w#PSCInitY+2*PSCHeight+20,4(a4) ; store y coord
move.w#PSRight,6(a4) ; store x coord

; Push parameters for NewControl
clri .(sp) ; NewControl returns a handle
move.l WDHAPSPtr,-(sp) ; the window ptr
lea TRect ; the rectangle bounding the control
lea 'Output Attenuation' ; title
move.b #TRUE,-(sp) ; visible
move.w #0,-(sp) : value
move.w #0,-(sp) : min
move.w #1,-(sp) : max
move.w #1,-(sp) ; check box proc id
move.w #0,-(sp) ; refcon not used

; Call NewControl

lea OAControl,a3
move.l (sp)+,-(a3) ; store the result

; Set up the controls bounding rectangle.
lea TRect,a4
move.w#PSChInitY+3*PSChHeight,(a4) ; store y coord
move.w#PSChInitX,2(a4) ; store x coord
move.w#PSChInitY+3*PSChHeight+20.4(a4) ; store y coord
move.w#PSRight,8(a4) ; store x coord

; Push parameters for NewControl
clr.l -(sp) ; NewControl returns a handle
move.l WDHAPSPtr,-(sp) ; the window ptr
lea TRect ; the rectangle bounding the control
lea 'Field Mike' ; title
move.b#TRUE,-(sp) ; visible
move.w#1,-(sp) ; make Field mike on as the default
move.w#O,-(sp) ; min
move.w#1,-(sp) ; max
move.w#2,-(sp) ; radio button proc id
move.w #O,-(sp) ; refcon not used

; Call NewControl

lea FieldControl,a3
move.l (sp)+,-(a3) ; store the result

; Set up the controls bounding rectangle.
lea TRect,a4
move.w#PSChInitY+4*PSChHeight,(a4) ; store y coord
move.w#PSChInitX,2(a4) ; store x coord
move.w#PSChInitY+4*PSChHeight+20.4(a4) ; store y coord
move.w#PSRight,8(a4) ; store x coord

; Push parameters for NewControl
clr.l -(sp) ; NewControl returns a handle
move.l WDHAPSPtr,-(sp) ; the window ptr
lea TRect ; the rectangle bounding the control
lea 'Probe Mike' ; title
move.b#TRUE,-(sp) ; visible
move.w#0,-(sp) ; value
move.w#0,-(sp) ; min
move.w#1,-(sp) ; max
move.w#2,-(sp) ; radio button proc id
move.w #O,-(sp) ; refcon not used

; Call NewControl

lea ProbeControl,a3
move.l (sp)+,-(a3) ; store the result
movem.l (sp)+,-(sp) ; store the result
CalThetaRect calculates the rectangle surrounding the control bar for the given channel.
Input: the channel # (a word) is passed on the stack.
Output: the rect TRect is filled.

CalThetaRect:

```
movem.l d0-d7/a0-a6,-(sp)  ; (sp)+, d0-d7/a0-a6
lea TRect,a4 ; get address of TRect
move.w #PSGIniY+PSGHeight,d4 ; bottom of graph
move.w d4,4(a4) ; store it in TRect
lea Theta0,a3 ; Get theta
move.w d4,4(sp),d3 ; Get channel number
asl.w #2,d3 ; '4
sub.w (a3,d3.w),d4 ; compute top of bar y coord
move.w d4,a4 ; store it in TRect
move.w d4,4(sp),d3 ; Get channel number
mulu #PSGChnWidth,d3 ; channel # * ChanWidth
add.w #PSGIniX,d3 ; move over
move.w d3,2(a4) ; store left side
add.w #PSGChnWidth,d3 ; add width
move.w d3,6(a4) ; store right side
lea TRect
movem.l (sp)+,d0-d7/a0-a6
movem.l d0-d7/a0-a6,-(sp)  ; (sp)+, d0-d7/a0-a6
lea TRect,a4
move.w #PSGIniY,d4 ; top of graph
move.w d4,4(a4) ; store it in TRect
lea Phi0,a3 ; Get Phi
move.w d4,4(sp),d3 ; Get channel number
asl.w #2,d3 ; '4
sub.w (a3,d3.w),d5 ; compute bottom of bar y coord
add.w d5,d4
move.w d4,4(a4) ; store it in TRect
move.w d4,4(sp),d3 ; Get channel number
mulu #PSGChnWidth,d3 ; channel # * ChanWidth
add.w #PSGIniX,d3 ; move over
move.w d3,2(a4) ; store left side
add.w #PSGChnWidth,d3 ; add width
move.w d3,6(a4) ; store right side
lea TRect
move.w #1,-(sp)
```

CalPhiRect calculates the rectangle surrounding the control bar for the given channel.
Input: the channel # (a word) is passed on the stack.
Output: the rect TRect is filled.

CalPhiRect:

```
movem.l d0-d7/a0-a6,-(sp)  ; (sp)+, d0-d7/a0-a6
lea TRect,a4
move.w #PSGIniY,d4 ; top of graph
move.w d4,4(a4) ; store it in TRect
lea Phi0,a3 ; Get Phi
move.w d4,4(sp),d3 ; Get channel number
asl.w #2,d3 ; '4
sub.w (a3,d3.w),d5 ; compute bottom of bar y coord
add.w d5,d4
move.w d4,4(a4) ; store it in TRect
move.w d4,4(sp),d3 ; Get channel number
mulu #PSGChnWidth,d3 ; channel # * ChanWidth
add.w #PSGIniX,d3 ; move over
move.w d3,2(a4) ; store left side
add.w #PSGChnWidth,d3 ; add width
move.w d3,6(a4) ; store right side
lea TRect
move.w #1,-(sp)
```
move.w #1,-(sp)
_InsetRect
: make it a tad smaller
add.w #1,4,(a4) ; not the bottom though
movem.i (sp)+,d0-d7/a0-a6
move.i (sp),2(sp) ; move return address over parameters
sub.w (sp)+ ; get rid of parameter
rts ; and return

; Name: PrintVal
; Function: This routine prints the given value at the specified row and
column of the PSChart.
; Input: d3 (word) = value, d4 = row, d5 = column
; Output: None
PrintVal:
movem.i d0-d7/a0-a6,-(sp) ; save registers
move.w #64,(sp),d3 ; d3 = value to be printed
move.w #66,(sp),d4 ; d4 = Row in chart
move.w #88,(sp),d5 ; d5 = column in chart
comp x coord
mulu #PSCFWidth,d1 ; column * width of each field
add.w #PSCInitX+Z,d5 ; shift over
comp y coord
add.w #1,d4 ; add 1 to row
mulu #PSCFHeight,d4 ; * height of each field
add.w #PSCInitY-6,d4 ; shift down and then up a little
erase whatever is there already.
leas TRect,a2 ; we'll put it in TRect
move.w #5,(a2) ; our x is the left x
move.w #6,(a2) ; then compute the right
add.w #20,(a2) ; as 20 over from the left
move.w #4,(a2) ; our y is the bottom y
move.w #4,(a2) ; then compute the top
sub.w #PSTxtSize,(a2) ; as TxtSize up from bottom
pea TRect ; now erase d
_EraseRect
move there
move.w #5,-(sp)
move.w #4,-(sp)
_MoveTo
convert value to string
move.w d3,d0 ; NumToString expects val in d0
lea NumBuf,a0 ; address of NumBuf in a0
move.w #0,(sp) ; Select NumToString
_Pack7
pea NumBuf
_DrawString
movem.i (sp)+,d0-d7/a0-a6
move.i (sp),6,(sp) ; move return address over parameters
add.i #6,sp ; get rid of parameters
rts

; Name: WDHAPSIS
; Function: This routine returns a Boolean telling whether or not
; the given window pointer is the PS window's pointer.
Input: A window pointer (passed on the stack)
Output: a word, TRUE or FALSE (defined in WDHA.hdr) returned on the stack.
**Note:** You do not have to push a word for the result of this routine.

**WDHAPSIS:**

```
movem.1 a4/d4,-(sp) ; save registers
move.1 #sp,a4 ; get return address in a4
move.1 12(sp),d4 ; get WindowPtr in d4
cmp.1 WDHAPSPtr,d4 ; Was it our window?
beq move.w IS10 ; It is
move.w #FALSE,14(sp) ; save result
brs IS20
IS10: move.w #TRUE,14(sp)
IS20: move.1 a4,10(sp) ; put return address back
movem.1 (sp)+,a4/d4 ; restore registers
tst.w (sp)+ ; get rid of extra two bytes
rts
```

Name: WDHAPSControl
Function: This routine should be called whenever a mouse down event occurs
within the contents of the PS Window. It handles the highlighting of the
proper control buttons, and sends the proper records to the WDHA.
Input: The mouse location (on the stack), from the event's where field.
Output: None

**WDHAPSConml:**

```
movem.1 d0-d7/a0-a6,-[sp) ; WDHAPSPtr on stack
move.1 WDHAPSPtr,-(sp) ; SetPort
PROCEDURE SetPort (sp: GralPor)
port

pea 64(sp) ; push address of point
_GlobalToLocal

; Was it in a control button?
ButtonCheck:
; call FindControl
clr.w -[sp)
cmove.1 66(sp),-[sp)
cmove.1 WDHAPSPtr,-[sp)
pws WhichControl
_FindControl
tst.w (sp)+ ; pop result
lea WhichControl,a4
clsr ChanCheck
	 ; if it was in a control, call TrackControl
cfr.w -(sp)
cmove.1 WhichControl,-(sp)
cmove.1 70(sp),-(sp)
cmove.1 0,-(sp)
_TrackControl
tst.w (sp)+ ; did they change the button?
beg NoChan
	 ; if not then leave
lea WhichControl,a4
```
move.l OACControl,d4
cmp.l (a4),d4
bne NotOA

; It was the output attenuation button so adjust the bar heights.
c1r.w d3 ; use d3 as a channel counter
lea Theta0,a3
GLoop11:
cmp.w #CHANNELS,d3
bne InvBut
c1r.w -(sp)
bsr GOUT
move.w(a3),d0
sub.w (sp),d0
move.wd0,(a3)
move.w2(a3),d1
sub.w (sp)+,d1
move.wd1,(a3)
lea 4(a3),a3
add.w #1,d3
bra CGLoop11

InvBut:
c1r.w -(sp) ; GetCtlValue returns a word
move.i OACControl,-(sp)
_GetCtlValue
move.w(sp),d3
not.w d3
and.w #1,d3
move.i WhichControl,-(sp)
_SetCtlValue
move.wd3,-(sp)

CGLoop12:
cmp.w #CHANNELS,d3
bne UDScreen
c1r.w -(sp) ; get Theta in d0
add.w (sp),d0
move.wd0,(sp) ; add the new GOUT
move.wd0,-(sp) ; the new gain
bsr ValidGain
move.w(sp)+,a3
add.w (sp)+,d1
move.wd1,(sp) ; now clip the limit as necessary
move.wd1,-(sp) ; the new limit
bsr ValidLimit
move.w(sp)+,2(a3)
lea 4(a3),a3
add.w #1,d3 ; store phi
; It was the input attenuation button so adjust the bar heights.
clr.w d3 ; use d3 as a channel counter
lea Theta0,a3

CLoop21:
cmp.w #CHANNELS,d3
beq InvBut2
clr.w -(sp)
bsr GIN ; the gain (the limit is not affected)
move.w(a3),d0
sub.w (sp)+,d0
move.wd0,(a3)
; go to the next channel
lea 4(a3),a3
add.w #1,d3
bra CLoop21

InvBut2:
cmp.w -(sp)
move.w IAControl,-(sp)
_GetCtlValue
move.w(sp)+,d3
movw.d3
and.w #1,d3
move.w WhichControl,-(sp)
_SetCtlValue
clr.w d3 ; GetCtlValue returns a word
lea Theta0,a3

CLoop22:
cmp.w #CHANNELS,d3
beq UDScreen
clr.w -(sp)
bsr GIN ; get theta.
move.w(a3),d0
add.w (sp)+,d0
move.wd0,(sp) ; add the new GIN
move.wd3,(sp) ; now clip the gain as necessary
bsr ValidGain
move.w(sp)+,(a3) ; the new gain
; go to the next channel
lea 4(a3),a3
add.w #1,d3
bra CLoop22

UDScreen
bsr WCHAPSDraw
bra NoChan

; invert the control value
OtherBut:
cir.w -(sp)
mov.l WhichControl,-(sp)
_GetCtlValue
move.w(sp)+,d3
not.w d3
and.w #1,d3
mov.e WhichControl,-(sp)
move.wd3,-(sp)
_SetCtlValue

; Was it the Field button?
mov.e FieldControl,d4
lea WhichControl,a4
cmp.l (a4),d4
bne NotField

; Otherwise invert the Probe mike
clr.w -(sp)
mov.e ProbeControl,-(sp)
_GetCtlValue
move.w(sp)+,d3
not.w d3
and.w #1,d3
mov.e ProbeControl,-(sp)
move.wd3,-(sp)
_SetCtlValue
bra NoChan

; Was it the Probe button?
NotField:
mov.e ProbeControl,d4
lea WhichControl,a4
cmp.l (a4),d4
bne NoChan

; Otherwise invert the Field mike
clr.w -(sp)
mov.e FieldControl,-(sp)
_GetCtlValue
move.w(sp)+,d3
not.w d3
and.w #1,d3
mov.e FieldControl,-(sp)
move.wd3,-(sp)
_SetCtlValue
bra NoChan

ChanCheck:
mov.w #0,d4
lea Theta0,a4
FindChan:
cmp.w #CHANNELS,d4
beq NoChan

75
move.wd4,-(sp) 
bsr CalThetaRect ; Calculate theta rectangle 
cir.w -(sp) ; make room for result
move.l 66(sp),-(sp) ; push mouse point
pea TRect ; theta rect in TRect
_plinRect

tst.w (sp)+
bne FoundTheta

; Is it a phi bar?
lea 2(a4),a4
move.wd4,-(sp)
bsr CalPhiRect ; Calculate theta rectangle 
cir.w -(sp) ; make room for result
move.l 66(sp),-(sp) ; push mouse point
pea TRect ; theta rect in TRect
_plinRect

tst.w (sp)+
bne FoundPhi

lea 2(a4),a4
add.w #1,d4
bra FindChan

: a4 points to Theta, d4 contains the channel number.

FoundTheta:

pea ThetaPat
  _PenPat
  move.w(a4),d3 ; hold onto original theta

; While the button is down move the bar around, changing theta
PLoop:
cir.w -(sp) ; Make room for result
  _StillDown
  tst.w (sp)+
bnez NoChan ; If not then exit otherwise...

; Get the point
pea TPoint
  _GetMouse

; First Erase Old Bar
move.w*patBlc,-(sp)
  _PenMode
move.wd4,-(sp)
bsr CalThetaRect
pea TRect
_paintRect

; Now change the theta parameter
move.w64(sp),d5 ; the vertical coordinate of start point
suo.w TPoint,d5 ; original y - current y

; This will be a negative value if they move down
move.wd3,(a4) ; restore original theta
add.w d5,(a4) ; change theta

; Is it OK?
move.wd4,-(sp) ; channel #
move.w(a4),-(sp) ; gain
bsr ValidGain ; make sure gain is in range
move.w(sp)+,(a4)
; Now draw the new bar
THDrBar:
    move.w #patOr,-(sp)
    _PenMode
    move.w #4,-(sp)
    bsr CalThetaRect
    pea TRect
    _PaintRect
; Now update the chart value.
    cmp.w (a4),#3 : is there any difference?
    beq FLoop
; If not then don't bother
    move.w #(1,-(sp)) ; move the value in to the chart
    bsr CalThetaRect
    pea TRect
    _PaintRect
    move.w (a4), -(sp) : value
    bsr PrintVal
    bra FLoop

; a4 points to Phi, d4 contains the channel number.
FoundPhi:
    pea PhiPat
    _PenPat
    move.w (a4),#3 ; store old Phi

; While the button is down move the bar around, changing theta
FPLoop:
    clr.w -(sp) ; Make room for result
    _StillDown
    tst.w -(sp) ; Is the button still down?
    beq NoChan ; If not then exit otherwise...
    Get the point
    pea TPoint : Get mouse location
    _GetMouse

; First Erase Old Bar
    move.w #patBlc,-(sp)
    _PenMode
    move.w #4,-(sp)
    bsr CalPhiRect
    pea TRect
    _PaintRect
; Now change the Phi parameter
    move.w#64,(sp),#5 ; the vertical coordinate of start point
    sub.w TPPoint,(sp) ; original y - current y
    ; this will be a negative value if they move down
    move.w (a4),#5 ; restore original Phi
    add.w d5,(a4) ; change Phi
    ; is it OK?
    move.w (sp),#4 ; channel #
    move.w (a4),#4 ; limit
    bsr ValidLimit ; make sure limit in range
    move.w (sp),#4

; Now draw the new bar
PhiDrBar:
; Now draw the new bar
    move.w #patOr,-(sp)
5,706,352

PenMode
move.w,4-,(sp)
bar CalPhiRed
lea TRect
PaintRect

Now update the chart value.

cmp.w, (a4),d3 ; is there any difference?
bq FLoop ; If not then don't bother
move.w#2,-(sp) ; limit column in chart
move.w#4,-(sp) ; row is channel #
add.w #1,(sp)+1
move.w(a4),-(sp) ; value
bar PrintVal
bra FLoop

NoChan:

PenNormal
bar WDHAPSSetParam ; update any changes made to the WDHA.
move.l (sp)+,d0-d7/a0-a6
move.l (sp)+,(sp) ; get rid of param
rts

; Name: WDHAPSSetParam
; Function: This routine sets the WDHA to the parameters set in the WDHA
; window.
; Input: None
; Output: None

WDHAPSSetParam:

move.l d0-d7/a0-a6,-(sp) ; save registers
bar CalcGainsLimits ; calculate the gains and limits.

; Now calculate the select word by looking at the control buttons.
lea paramrec,a4 ; get the gain/input select word
move.w16(a4),d4 ; get the gain input select word

SPIA:

clr.w -(sp) ; GetCtlValue returns a word
move.l IAControl,-(sp) ; the handle
GetCtlValue
tst.w (sp)+
beq SPNoIA

SPDolA:
bset.l #INPUT,d4
bra SPOA

SPNoIA:
bclr.l #INPUT,d4

SPOA:

clr.w -(sp) ; GetCtlValue returns a word
move.l OAControl,-(sp) ; the handle
GetCtlValue
tst.w (sp)+
beq SPNoOA

SPDolOA:
bset.l #OUTPUT,d4
bra SPFila

SPNoOA:
bcir.l #OUTPUT.d4

SPField:
  clr.w -(sp) ; GetCtlValue returns a word
  move.i FieldControl-(sp) ; the handle
  _GetCtlValue
  tst.w (sp)+
  beq SPNoField

SPDoField:
  bset.i #FIELD,d4
  bra SPProbe

SPNoField:
  bcir.l #FIELD,d4

SPProbe:
  clr.w -(sp) ; GetCtlValue returns a word
  move.i ProbeControl-(sp) ; the handle
  _GetCtlValue
  tst.w (sp)+
  beq SPNoProbe

SPDoProbe:
  bset.i #PROBE,d4
  bra SPSendParams

SPNoProbe:
  bcir.l #PROBE,d4

SPSendParams:
  move.wd4,16(a4) ; store the modified select word.

; Now send the parameters to the WDHA
lea paramrac,a0
bsr SetParam

; now wait a little while while the WDHA does it's thing.
move.l #10000,d1

SPWait:
  sub.l #1,d1
  bra SPWait

; Now put the WDHA in either hearing aid state or idle state depending on
; the status of the "Hearing Aid On" button.
  clr.w -(sp) ; GetCtlValue returns a word
  move.i AidControl-(sp) ; the handle
  _GetCtlValue
  tst.w (sp)+
  beq SPAidOff

move.w #1,d0 ; go to hearing aid mode
bra SPSendMode

SPAidOff:
  move.w #100, d0 ; go to idle mode

SPSetMode:
  jsr scsiwr ; send mode code to WDHA

SPDone:
  movem.l (sp)+,d0-d7/a0-a6 ; restore registers

; Name: CalcGainsLimits
; Function: Compute the gains and limits fields of the paramrec from
the heights of the theta and phi bars of the bar graph, and the status of
; the attenuation control buttons.
; Input: None
; Output: None
; If any of the gains or limits produce an out of range value the
; variable called 'Clipped' will have a non-zero value upon return.

CalcGainsLimits:
  movem.1 a0-a8/d0-d7,-(sp)
  lea Clipped,a1
  clr.w (a1)
  lea Theta0,a4 ; theta0 here
  lea paramrec,a2 ; gain0 here
  lea He,a3
  move.w CHANNELS,d6 ; loop through four channels

DCLoop:
  move.w(a4),d4 ; get theta0 (= So)
  sub.w (a3),d4 ; subtract He
  sub.w #60,d4 ; subtract Hr
  sub.w 8(a3),d4 ; subtract GIN
  clr.w -(sp) ; subtract GOUT
  bsr GIN
  sub.w (sp)+.d4 ; subtract GOUT

  DCLoop:

  DoLimit:
  move.w2(a4),d5 ; Get height (=So lim) in d5
  sub.w d4,d5 ; Subtract Gd
  sub.w 8(a3),d5 ; subtract Hr
  clr.w -(sp) ; subtract GOUT
  bsr GOUT
  sub.w (sp)+.d5

  ToLinear:

  but first store Gd and Ld
  move.w d4,(a0) ; store Gd
  move.w d5,(a6) ; store Ld
  lea arg1,a0
  move.w d4,(a0) ; store gain (dB) in arg1
  peax arg1 ; dB gain
  peax arg4 ; fpdB gain
  fp2dB arg4 ; convert from integer to extended fp
  peax fp2dB arg4 ; store gain (db) in arg4
  fabs arg4 ; base e exponential (db ratio in arg4)
  fpex arg4 ; scale it *2E18 to convert it to fixed point
  lea arg4
  peax arg4
  peax arg1

  80
; Now the limit
DCDoLimit:
lea arg1,a0
move.w d5,(a0) ; store limit (dB) in arg1
lea arg1 ;dB limit
lea arg4 ;fDB limit
f2X ;convert from integer to extended fp
lea lp20dBBe ; 20 * log base 10 of e = 8.68588638
lea arg4 ;fDB limit
fx2i ;convert extended to integer
move.w arg1,2(a2) ; store the gain
add.w #1,(a1)
lea arg4
lea arg4
lea arg1
lea twoex14 ;scale it *2E16 to convert it to fixed point
lea arg4
fx2i ;convert extended to integer
move.w arg1,2(a2) ; store the limit
bpl DCFinLoop
move.w #32767,2(a2) ;convert from integer to extended fp

; Store them in the paramrec
DCFinLoop:
lea 4(a4),a4 ; go to next theta/phi pair
lea 4(a2),a2 ; go to next gain/limit pair
lea 2(a3),a3 ; go to next He and Hr
subq.b #f,d6
bne DCLoop
movem.1 (sp)+,a0-a6/d0-d7
rts

; Name: GIN
; Function: This routine returns the input gain as determined by the
; input attenuation control button, either +0 (on), or -18 (off).
; Input: None
; Output: A word on the stack is filled with the result (the user pushes this)
GIN: movem.1 a0-a6/d0-d7,-(sp)
; if input attenuation is on then return 0 otherwise -18
clr.w -(sp) ; make room for result
move.1 IAControl,-(sp)
_GetCtlValue
lst.w (sp)+
bra GinOn
move.w #18,64(sp)
bra GinDone
GinOn
move.w#0,64(sp)

GoutDone

move.w (sp)+,d0-a6/a0-64(sp)

ts

; Name: GOUT
; Function: This routine returns the output gain as determined by the
; output attenuation control button, either -34 (on), or -9 (off).
; Input: None
; Output: A word on the stack is filled with the result (the user pushes this)
GOUT: movem.l a0-a6/d0-d7,-(sp)

if output gain is on then return -34 otherwise -9

c1r.w -(sp) ; make room for result
move.l OACControl,-(sp)
_GetICValue
fst.w (sp)+

bne GoutOn
move.w#-9,64(sp)

bra GoutDone

GoutOn

move.w#-34,64(sp)
GoutDone

movem.l (sp)+,a0-a6/d0-d7

rts

; Name: GMAX
; Function: This routine returns the maximum gain for the given channel.
; Input: The channel number is passed on the stack as a word (0-3).
; Output: The result is on top of the stack upon return.
; ***Note: You do not have to make room for the result on the stack.
GMAX:

movem.l a0-a6/d0-d7,-(sp)
move.w60,d0; hold result in d0

c1r.w -(sp)

brr

add.w (sp)+,d0 ; add GIN
c1r.w -(sp)

bar

add.w (sp)+,d0 ; add GOUT
lea H,e,0

move.w64(sp),d1 ; get channel #

asl.w #1,a0 ; 2 for words

add.w (a0,d1),d0 ; add He
add.w #a0,a0,d1.w,d0 ; add Hr

move.w64(sp) ; write the result over the parameter

movem.l (sp)+,a0-a6/d0-d7

rts

; Name: ValidGain
; Function: This routine clips the given gain (bar height) as needed for the
; given channel.
; Input: The channel number and gain passed on the stack as words.
; Output: The result is on top of the stack upon return.
; ***Note: You do not have to make room for the result on the stack.
ValidGain:
  movem.l a0-a6/d0-d7,-(sp)
  move.w66(sp),d0 ; get the channel #
  move.w44(sp),d1 ; get the unclipped gain
  cmp.w #2,d1 ; IS it bigger than the minimum height?
  bge GainOK1 ; make it bigger
  bra VGDone

GainOK1:
  move.w0,-(sp) ; get GMAX
  bsr GMAX
  cmp.w (sp)+,d1
  ble VGDone ; make it GMAX

VGDone:
  move.w1,66(sp)
  movem.l (sp)+,a0-a6/d0-d7
  move.l (sp)+,2(sp) ; move return address
  tst.w (sp)+ ; get rid of extra word
  rts

; Name: LMAX
; Function: This routine returns the maximum limit for the given channel.
; Input: The channel number is passed on the stack as a word (0-3).
; Output: The result is on the stack upon return.
; ***Note: You do not have to make room for the result on the stack.
LMAX:
  movem.l a0-a6/d0-d7,-(sp)
  clr.w -(sp)
  bsr GOUT
  move.w(sp)+,d0 ; add GOUT
  lea H,a0
  move.w44(sp),d1 ; get channel #
  asl.w #1,d1 ; *2 for words
  add.w (a0,d1.w),d0 ; add Hr
  move.w0,64(sp) ; write the result over the parameter
  movem.l (sp)+,a0-a6/d0-d7
  rts

; Name: ValidLimit
; Function: This routine clips the given limit (bar height) as needed for the
given channel.
; Input: The channel number and gain passed on the stack as words.
; Output: The result is on top of the stack upon return.
; ***Note: You do not have to make room for the result on the stack.
ValidLimit:
  movem.l a0-a6/d0-d7,-(sp)
  move.w66(sp),d0 ; get the channel #
  move.w44(sp),d1 ; get the unclipped limit
  cmp.w #2,d1 ; IS it bigger than the minimum height?
  bge LimitOK1 ; make it bigger
  bra VLDone

LimitOK1:
move.w-0,(sp) ; get LMAX

section

movem.l (sp)+,a0-a5/d0-d7
tst.w (sp)+ ; get rid of extra word

rets

:-------------WDHAPS data declarations---------------------

align 2 ; align to word boundary
Theta0: DC.W 50
Phi0: DC.W 70
Theta1: DC.W 50
Phi1: DC.W 70
Theta2: DC.W 50
Phi2: DC.W 70
Theta3: DC.W 50
Phi3: DC.W 70

paramrec: ; WDMA parameter record
dc.w 16384 ; channel 0 gain
dc.w 16384 ; channel 1 gain
dc.w 16384 ; channel 2 gain
dc.w 16384 ; channel 3 gain
dc.w 4224 ; gain/input select word

He: ; The He table must(!) follow the He table.
dc.w 121 ; channel 0
dc.w 117 ; channel 1
dc.w 127 ; channel 2
dc.w 120 ; channel 3

WDHAPSBounds: ; Bounding rect for window
  DC.W PSLinitY
  DC.W PSLinitX
  DC.W PSLinitY+PSGHeight+PSGinitY+2*PSTxSize+4
  DC.W PSLRight

WDHAPSGraph: ; bounding rectangle for graph
  DC.W PSGInitY
  DC.W PSGinitX
  DC.W PSGinitY+PSGHeight
  DC.W PSGinitX+PSGWidth

WDHAPSChart: ; bounding rectangle for chart
  DC.W PSGInitY
  DC.W PSGinitX
  DC.W PSGinitY+PSGHeight
  DC.W PSGinitX+PSGWidth

TRect:
  DCL 0
  DCL 0 ; For calculating various rectangles.

TPoint:
  DCL 0 ; For calculating mouse change.

WhichControl: DCL 0 ; A control handle, for temporary storage.

ThetaPat: DCB $AA,$55,$AA,$55,$AA,$AA,$55
PhiPat: DGB $55,$AA,$55,$AA,$55,$AA,$AA

NumBuf: DCB.B 64.0 ; Buffer for number conversion
arg1 dcb.w 8.0 ; integer buffer
arg2 dcb.w 8.0 ; extended floating point buffer
arg3 dcb.w 8.0 ; extended floating point buffer
arg4 dcb.w 8.0 ; extended floating point buffer
arg5 dcb.w 8.0 ; extended floating point buffer
twox14 dc.w 400.d,$8000,$0000,$0000,$0000
lp20dBe dc.w 400.d,$8009,$6d22,$30e6,$8042

Clipped dc.w 0
This file must be included if your program uses the WDHA Parameter Settings window.

XREF WDHAPSOpen
XREF WDHAPSClose
XREF WDHAPSShow
XREF WDHAPSHide
XREF WDHAPSDraw
XREF WDHAPSControl
XREF WDHAPSSet
XREF WDHAPSSetParam
; file WDHA5.C.Asm
Include MacTraps.D
Include ToolEqn.D
Include SystEqn.D
Include QuickEqn.XD
Include SANEMac.txt
Include MDS2:WDHA.hdr
Include MDS2:WDHArMac.txt
Include MDS2:WDHASC5I.hdr

; WDHA Test/Calibrate Window Manager
; This package contains routines to manipulate the WDHA Test/Calibrate
; window, which allows you to do pure tone audiometry via the WDHA.
; The window contains text boxes which allow the user to change the
; parameters to the test procedure, as well as the control boxes (as in the
; parameter settings window) to determine the gain/select input word and
; the on/off status of the hearing aid.

; --------------------- External Definitions ---------------------
XDEF WDHA5Open
XDEF WDHA5Close
XDEF WDHA5Show
XDEF WDHA5Hide
XDEF WDHA5Draw
XDEF WDHA5Control
XDEF WDHA5Dial
XDEF WDHA5Key
XDEF WDHA5Get
XDEF WDHA5DoTest

; --------------------- Constant Definitions ---------------------
; TC = The Test/Calibrate Window
TCInitX EQU 30 ; initial X coord (global) of upper left corner
TCInitY EQU 50 ; initial Y coord (global) of upper left corner
TCRight EQU 448
TCTextSize EQU 12

; TCC = The Control Buttons
TCCInitX EQU 258
TCCInitY EQU 15
TCCHeight EQU 24

; Text Edit Box Constants
ToneBursts EQU 0
RiseCount EQU 1
OnCount EQU 2
FallCount EQU 3
OffCount EQU 4
Frequency EQU 5
Attenuate EQU 6
There are seven boxes.

--- Subroutine Declarations ---

: Name: WDHATCOpen
: Function: Call this routine to create and display the TC Window.
: Input: None
: Output: None

WDHATCOpen:
  movem.1 d0-d2/a0-a6,-(sp) ; save registers
  movem.1 d0-d7/a0-a6,-(sp) ; save registers
  movem.1 d0-d7/a0-a6,-(sp) ; save registers

; Set up document window.
  : FUNCTION NewWindow [wStorage: Ptr; boundsRect: Rect;
  : title: Str255; visible: BOOLEAN;
  : procID: INTEGER; behind: WindowPtr;
  : goAwayFlag: BOOLEAN;
  : relCon: LongInt] : WindowPtr;
  SUBQ #4,SP ; Space for function result
  CALL .-(SP) ; Storage for window (Heap)
  PEA WDHATCBounds ; Window position
  PEA WDHATCTitle ; Window title
  MOVE #255,.(SP) ; Make window visible
  MOVE #rDocProc,.(SP) ; Standard document window
  move.B #1,.(SP) ; Make it the front window
  CURL .-(SP) ; Window has goAway button
  lea WDHATCPtr.a4 ; Create and draw window
  MOVE (SP)+,(a4) ; Save handle for later
  MOVEL (a4),.(SP) ; Make sure the new window is the port

: PROCEDURE SetPort (gp: GrafPort)
  ; Make it the current port
  br TCAddBoxes
  ; Add the control buttons.
  br TCAddControls
  ; Draw the content region
  br WDHATCDraw

  ; Name: WDHATCClose
  ; Function: Call this routine to destroy the TC Window and remove it from
  ; the screen.
  ; Input: None
  ; Output: None

WDHATCClose:
  movem.1 d0-d7/a0-a6,-(sp) ; save registers
  movem.1 WDHATCPtr,.- (sp)
  _KillControls

  ; Dispose Window
  movem.1 WDHATCPtr,.- (sp)
  _DisposeWindow
  movem.1 (sp)+,d0-d7/a0-a6 ; restore registers
  rts
WDHATCShow:

; Function: This routine makes the TC window visible and frontmost.
; Input: None
; Output: None

 WDHATCShow:

movem.1 d0-d7/a0-a6,-(sp) ; save registers
  ; Bring it to the front
  move.1 WDHATCPtr,-(sp)
  _BringToFront
  ; Show Window
  move.1 WDHATCPtr,-(sp)
  _ShowWindow
  move.1 WDHATCPtr,-(sp)
  _SelectWindow
  movem.1 (sp)+,d0-d7/a0-a6 ; restore registers
  rts

WDHATCHide:

; Function: This routine makes the TC window invisible, removing it from the
; screen (but not destroying it).
; Input: None
; Output: None

 WDHATCHide:

movem.1 d0-d7/a0-a6,-(sp) ; save registers
  ; Hide Window
  move.1 WDHATCPtr,-(sp)
  _HideWindow
  movem.1 (sp)+,d0-d7/a0-a6 ; restore registers
  rts

WDHATCDraw:

; Function: This routine draws the TC window's contents.
; Input: None
; Output: None

 WDHATCDraw:

movem.1 d0-d7/a0-a6,-(sp) ; save registers
  lea WDHATCPtr,a4 ; Pointer on stack
  MOVEL (a4),-(SP)
  ;PROCEDURE SetPort (gp: GrafPort)
  _SetPort ; Make it the current port
  ; Draw the text buttons.
  bsr TCDrawBoxes
  ; Draw the control buttons.
  move.1 WDHATCPtr,-(sp) ; the window ptr
  _DrawControls
  movem.1 (sp)+,d0-d7/a0-a6 ; restore registers
  rts

TCAddControls:

; Function: This routine adds the TC window's controls.
; Input: None
; Output: None

 TCAddControls:

movem.1 d0-d7/a0-a6,-(sp) ; save registers
Set up the controls bounding rectangle.

lea TRect,a4
move.w#TCCInitY+1^TCCLFHeight,(a4) ; store y coord
move.w#TCCInitX,2(a4) ; store x coord
move.w#TCCInitY+1^TCCLFHeight+20,(a4) ; store y coord
move.w#TCRight,6(a4) ; store x coord

; Push parameters for NewControl
clr.1 -(SP) ; NewControl returns a handle
move.l WDHTCPtr,-(sp) ; the window ptr
lea TRect ; the rectangle bounding the control
lea "Hearing Aid On" ; title
move.b #TRUE,-(sp) ; visible
move.w#0,-(sp) ; value
move.w#0,-(sp) ; min
move.w#1,-(sp) ; max
move.w#1,-(sp) ; check box proc id
move.l #0,-(sp) ; refcon not used

; Call NewControl
_newControl
lea IAControl,a3
move.l (sp)+,(a3) ; store the result

; Set up the controls bounding rectangle.
lea TRect,a4
move.w#TCCInitY+1^TCCLFHeight,(a4) ; store y coord
move.w#TCCInitX,2(a4) ; store x coord
move.w#TCCInitY+1^TCCLFHeight+20,(a4) ; store y coord
move.w#TCRight,6(a4) ; store x coord

; Push parameters for NewControl
cIr.1 -(sp) ; NewControl returns a handle
lea TRect ; the rectangle bounding the control
lea "Input Attenuation" ; title
move.b #TRUE,-(sp) ; visible
move.w#0,-(sp) ; value
move.w#0,-(sp) ; min
move.w#1,-(sp) ; max
move.w#1,-(sp) ; check box proc id
move.l #0,-(sp) ; refcon not used

; Set up the controls bounding rectangle.
lea TRect,a4
move.w#TCCInitY+2^TCCLFHeight,(a4) ; store y coord
move.w#TCCInitX,2(a4) ; store x coord
move.w#TCCInitY+2^TCCLFHeight+20,(a4) ; store y coord
move.w#TCRight,8(a4) ; store x coord

; Push parameters for NewControl
cIr.1 -(sp) ; NewControl returns a handle
lea TRect ; the rectangle bounding the control
lea "Output Attenuation" ; title
move.b #TRUE,-(sp) ; visible
move.w #0,-(sp) ; value
move.w #0,-(sp) ; min
move.w #1,-(sp) ; max
move.w #1,-(sp) ; check box proc id
move.l #0,-(sp) ; refcon not used

; Call NewControl
    NewControl
lea OAControl, a3
move.l (sp)+,(a3) ; store the result
lea TRect, a4
move.w #TCCtlInitY+3*TCCtlFHeight,(a4) ; store y coord
move.w #TCCtlInitX,2(a4) ; store x coord
move.w #TCCtlInitY+3*TCCtlFHeight+20,4(a4) ; store y coord
move.w #TCCtlRight,6(a4) ; store x coord

; Push parameters for NewControl
cr.l -(sp) ; NewControl returns a handle
move.l WDHATCPtr,-(sp) ; the window ptr
pea TRect ; the rectangle bounding the control
pea 'Field Mike' ; title
move.b #TRUE,-(sp) ; visible
move.w #TCCtlInitY+4*TCCtlFHeight,(a4) ; store x coord
move.w #TCCtlInitX+4*TCCtlFHeight+20,4(a4) ; store y coord
move.w #TCCtlRight,5(a4) ; store x coord

; Set up the controls bounding rectangle.
lea TRect, a4
move.w #TCCtlInitY+3*TCCtlFHeight,(a4) ; store y coord
move.w #TCCtlInitX,2(a4) ; store x coord
move.w #TCCtlInitY+3*TCCtlFHeight+20,4(a4) ; store y coord
move.w #TCCtlRight,6(a4) ; store x coord

; Push parameters for NewControl
cr.l -(sp) ; NewControl returns a handle
move.l WDHATCPtr,-(sp) ; the window ptr
pea TRect ; the rectangle bounding the control
pea 'Probe Mike' ; title
move.b #TRUE,-(sp) ; visible
move.w #0,-(sp) ; value
move.w #0,-(sp) ; min
move.w #1,-(sp) ; max
move.w #2,-(sp) ; radio button proc id
move.l #0,-(sp) ; refcon not used

; Call NewControl
    NewControl
lea FieldControl, a3
move.l (sp)+,(a3) ; store the result
lea TRect, a4
move.w #TCCtlInitY+3*TCCtlFHeight,(a4) ; store y coord
move.w #TCCtlInitX,2(a4) ; store x coord
move.w #TCCtlInitY+3*TCCtlFHeight+20,4(a4) ; store y coord
move.w #TCCtlRight,6(a4) ; store x coord

; Push parameters for NewControl
cr.l -(sp) ; NewControl returns a handle
move.l WDHATCPtr,-(sp) ; the window ptr
pea TRect ; the rectangle bounding the control
pea 'Probe Mike' ; title
move.b #TRUE,-(sp) ; visible
move.w #0,-(sp) ; value
move.w #0,-(sp) ; min
move.w #1,-(sp) ; max
move.w #2,-(sp) ; radio button proc id
move.l #0,-(sp) ; refcon not used

; Call NewControl
    NewControl
lea ProbeControl, a3
move.l (sp)+,(a3) ; store the result
lea TRect, a4
move.w #TCCtlInitY+5*TCCtlFHeight,(a4) ; store y coord
move.w #TCCInitX,2(a4) ; store x coord
move.w #TCCInitY+5*TCCHeight+24,4(a4) ; store y coord
move.w #TCCInitX+40,6(a4) ; store x coord

; Push parameters for NewControl

; NewControl returns a handle

; the window ptr

; the rectangle bounding the control

; title

; visible

; value

; min

; max

; simple button proc id

; rcon not used

; Call NewControl

lea StartControl,a3
movem.l (sp)+,(a3) ; store the result

TCAddBoxes:

movem.l d0-d7/a0-a6, -(sp)
lea TextHandles,a3
lea TextRects,a4
move.w #TextBoxes,d4

TCABLoop:

cmp.w #TextBoxes,d4
beq TCABDone

; TENew

; Get Destination Rect in TRect

; view rect

; dest rect

; make room for handle result

lea TRect,a2
movem.l (a4),(a2)

lea (a4),a2

move.w #1,(-sp)
move.w #1,-(sp)

InsetRect

; Call TENew

; TENew

lea TextHandles,a4

; Default Tone Burst is 3

lea '3',

; move past the length

lea #1,(-sp)

; It's 1 character long
move.1 (a4)+,-(sp)   
_TElInsert
; Default Rise Time is 309
  peal "309"            ; incorporate the text
  add.l 1,(sp)          ; move past the length
  move.1 #3,-(sp)       ; It's 3 characters long
  move.1 (a4)+,-(sp)    
_TElInsert
; Default Signal On is 2455
  peal "2455"           ; incorporate the text
  add.l #1,(sp)         ; move past the length
  move.1 #4,-(sp)       ; It's 4 characters long
  move.1 (a4)+,-(sp)    
_TElInsert
; Default Fall Time is 309
  peal "309"            ; incorporate the text
  add.l 1,(sp)          ; move past the length
  move.1 #3,-(sp)       ; It's 3 characters long
  move.1 (a4)+,-(sp)    
_TElInsert
; Default Signal Off is 3069
  peal "3069"           ; incorporate the text
  add.l 1,(sp)          ; move past the length
  move.1 #4,-(sp)       ; It's 4 characters long
  move.1 (a4)+,-(sp)    
_TElInsert
; Default Frequency is 2000
  peal "2000"           ; incorporate the text
  add.l 1,(sp)          ; move past the length
  move.1 #4,-(sp)       ; It's 4 characters long
  move.1 (a4)+,-(sp)    
_TElInsert
; Default Attenuation is 20
  peal "20"             ; incorporate the text
  add.l 1,(sp)          ; move past the length
  move.1 #2,-(sp)       ; It's 2 characters long
  move.1 (a4)+,-(sp)    
_TElInsert
  movem.l (sp)+,d0-d7/a0-a6
  rts

; Name: WDHATCIdle
; Function: This routine blinks the caret of the active text box. It should be
; called each time through your main event loop.
; Input: None
; Output: None

WDHATCIdle:             
  movem.l a0-a6/d0-d7,-(sp)
  lea TextHandles,a4
  move.w WActive,d4       ; which one is active?
  bmi TCINoneActive       ; -1 means none
  asl.w #2,d4             ; *4 for long offset
  move.1 (a4,d4,w),-(sp) 
  _TEIdle
TCINoneActive:
    movem.i (sp)+,a0-a6/d0-d7
    rts

; Name: WDHATKey
; Function: Call WDHATKey when the TC window is active and a keypress
; event is active.
; Input: The char (from the event's message field) as a word.
; Output: None

WDHATKey:
    movem.i a0-a6/d0-d7-(sp)
    lea TextHandles,a4
    move.w WWActive,d4 ; which one is active?
    bmi TCKNoneActive ; -1 means none
    esl.w #2,d4
    move.w 64(sp)-,(sp) ; push the char
    move.l (a4,d4.w)-(sp)

_TEKey
TCKNoneActive:
    movem.i (sp)+,a0-a6/d0-d7
    ; remove parameter from stack
    move.l (sp),2(sp)
    clr.w (sp)+ ; remove extra space
    rts

; Name: WDHATCIS
; Function: This routine returns a Boolean telling whether or not
; the given window pointer is the TC window's pointer.
; Input: A window pointer (passed on the stack)
; Output: a word, TRUE or FALSE (defined in WDHA.hdr) returned on the stack.
; "Note: You do not have to push a word for the result of this routine.

WDHATCIS:
    movem.l a4/d4,-(sp) ; save registers
    movem.l 12(sp),a4 ; get return address in a4
    move.l WWActive,d4 ; get WindowPtr in d4
    cmp.l WDHATCPtr,d4 ; Was it our window?
    beq IS10 ; It is
    move.w #FALSE,14(sp) ; save result
    bra IS20

IS10:
    move.w #TRUE,14(sp)

IS20:
    movem.l (sp)+,a4/d4 ; restore registers
    tst.w (sp)+ ; get rid of extra two bytes
    rts

; Name: WDHATCControl
; Function: This routine should be called whenever a mousedown event occurs
; within the contents of the TC Window. It handles the highlighting of the
; proper control buttons, and sends the proper records to the WDHA.
; Input: The mouse location (on the stack), from the event's where field.
; Output: None

WDHATCControl:
movem.l  d0-d7/a0-a6,-(sp)
move.l  WDHATCPr,-(sp)

;PROCEDURE  SetPort (gp: GrafPort)
port  _SetPort

pea  64(sp)
_GlobalToLocal

; Was it in a control button?

ButtonCheck:
; call FindControl
clr.w -(sp)
move.l  66(sp),-(sp)
pea  WhichControl
_FindControl
tst.w (sp)+
lea  WhichControl,a4
tsl.l  (a4)
beq  TBCheck

; if it was in a control, call TrackControl
clr.w -(sp)
move.l  70(sp),-(sp)
move.l  #0,-(sp)
_TrackControl
tst.w (sp)+
beq  NoChan

; Was it the Start Button?
move.l  StartControl,d4
lea  WhichControl,a4
cmp.l  (a4),d4
bne  InvControl
bra  WDHATCDoTest
bra  NoChan

; invert the control value

InvControl:
clr.w -(sp)
move.l  WhichControl,-(sp)
_GetCtlValue
move.w(sp)+,d3
not.w d3
and.w #1,d3
move.l  WhichControl,-(sp)
move.wd3,-(sp)
_SetCtlValue

; Was it the Field button?
move.l  FieldControl,d4
lea  WhichControl,a4
cmp.l  (a4),d4
bne  NotField

; Otherwise invert the Probe mke
clr.w -(sp)
move.l  ProbeControl,-(sp)

; Make sure it’s the current:

pea  64(sp)
_GlobalToLocal

; push address of point
; convert it to the window’s coords

; returns a long
; push point in local coords
; WDHATCPr on stack
; which one?
; pop result
; Was it in any of them?
; if not try the text boxes

; returns a word
; WhichControl now has the handle
; starting point
; no action proc
; did they change the button?
; if not then leave

; returns a word
; GetCtlValue returns a word
; now value is in d3
; invert the status
; set button

; if not then forget it
; GetCtlValue returns a word
; if not then forget it

; GetCtlValue returns a word
_GetCtlValue
move.w(sp)+,d3
not.w d3
and.w #1,d3
move.w ProbeControl,-(sp)
move.w d3,-(sp)

_SelCtlValue
bra NotChan

; Was it the Probe button?
NotField:
move.w ProbeControl,d4
lea WhichControl,a4
cmp.l (a4),d4
bne NoChan

; Otherwise invert the Field mike
clr.w -(sp)
move.w FieldControl,-(sp)
move.w d3,-(sp)
not.w d3
and.w #1,d3
move.w FieldControl,-(sp)
move.w d3,-(sp)

_SelCtlValue
bra NoChan

TBCheck:
lea TextRects,a4
move.w #ToneBurst,a4

TBLoop:
cmp.w #TextBoxes,d4
bne NoChan
clr.w -(sp)
move.w #Rect,(sp)
move.w a4,-(sp)
_PlnRect:
lea (sp)+

lea TextRects,a4
add.w #1,d4
bra TBLoop

TBFound:
lea TextHandles,a3
lea WAActive,a4
move.w a4,d3
bmi TBNActive
lea.w #2,d3
move.w a3,d3.w,-(sp)

TBNActive
move.w d4,(a4)
lea.w #2,d4
move.w a3,d4.w,-(sp)

_TEMActivate
}

; now value is in d3
; invert the status
; turn off Probe button
; GetCtlValue returns a word
; now value is in d3
; invert the status
; turn off Probe button
; GetCtlValue returns a word
; now value is in d3
; invert the status
; turn off Probe button
; if not then forget it
move.l 64(sp),-(sp) ; push the point
clr.w -(sp) ; don't extend
move.l (a3,d4.w),-(sp) ; push the TEHandle

_noClick

; Name: TCDrawBoxes
; Function: TCDrawBoxes draws the text box portion of the TC window, including the headings and the text boxes themselves.
; Input: None
; Output: None

TCDrawBoxes:
    movem.l d0,d7,a0-a6, -(sp)
pop ERect ; erase the input portion of the window
    lea TextRects,a4
    lea TextHandles,a3
    move.w #TCCtimerY+16,d3 ; initial y coord
    DispString #10,d3,Tone burst count?
pop 0(a4)
    _FrameRect
    pop ERect
    movem.1 0(a3),-(sp)
    _TEUpdate
    add.w #20,d3 ; move down
    DispString #10,d3,Rise time sample count?
pop 8(a4)
    _FrameRect
    pop ERect
    movem.l 4(a3),-(sp)
    _TEUpdate
    add.w #20,d3 ; move down
    DispString #10,d3,Signal on sample count?
pop 16(a4)
    _FrameRect
    pop ERect
    movem.1 8(a3),-(sp)
    _TEUpdate
    add.w #20,d3 ; move down
    DispString #10,d3,Fall time sample count?
pop 24(a4)
    _FrameRect
    pop ERect
    movem.1 12(a3),-(sp)
    _TEUpdate
    add.w #20,d3 ; move down
    DispString #10,d3,Signal off sample count?
pop 32(a4)
    _FrameRect
    pop ERect
move.l 18(a3),-(sp)
_TEUpdate
add.w #20, d3 ; move down
DispString #10, d3, Frequency?
pea 40(a4)
__FrameRect
pea ERect
move.1 20(a3), -(sp)
_TEUpdate
add.w #20, d3 ; move down
DispString #10, d3, Alten re max out (dB)
pea 48(a4)
__FrameRect
pea ERect
move.1 24(a3), -(sp)
_TEUpdate
add.w #20, d3 ; move down
DispValue #10, d3, Power = .PDcimal
pea
_DrawString
lea KeyBuf, a0
move.l PFract, d0
move.w #0, -(SP) ; Select NumToString
_Pack7
pea KeyBuf
_DrawString
movem.1 (sp)+, d0-d7/a0-a6
rts

: Name: WDHATCDoTest
: Function: WDHATCDoTest fills the paramrec with the proper values
: initiates the WDHA test by sending the paramrec out via the routine
: wdhatest.
: Input: None
: Output: None
WDHATCDoTest
movem.1 d0-d7/a0-a6, -(sp) ; save registers
lea paramrec, a4 ; generate the gain/input select word
: get the gain input select word
move.w 14(a4), d4
TCIA:
cir.w -(sp) ; GetCtlValue returns a word
move.l IAControl, -(sp) ; the handle
_GetCtlValue
tsl.w (sp)+
beq TCNoIA
TCDoA:
bsel.l #INPUT, d4
bra TCOA
TCNoIA:
bsel.l #INPUT, d4
TCOA:
cir.w -(sp) ; GetCtlValue returns a word
move.l OAControl, -(sp) ; the handle
_GetCtlValue

_lsl.4 (sp)+
beq TCNoOA

TCDoOA:

bset.l #OUTPUT.d4
bra TCFIELD

TCNoOA:

bclr.1 #OUTPUT.d4
bra TCFIELD

TCFIELD:

clr.w -(sp) ; GetCtlValue returns a word
move.l FIELDControl-(sp) ; the handle

_getCtlValue

_lsl.4 (sp)+
beq TCNoFIELD

TCDoField:

bset.l #FIELD.d4
bra TCFIELD

TCNoField:

bclr.1 #FIELD.d4
bra TCFIELD

TCPREME:

bclr.1 #FIELD.d4
bra TCFIELD

TCDoProbe:

bset.l #PROBE.d4
bra TCSendParams

TCNoProbe:

bclr.1 #PROBE.d4
bra TCSendParams

TCSendParams:

move.w,d4,14(a4) ; store the modified gain/input select word.

lea paramrec,a0
lea TCCVBoxes
lea whatcst:

move.l d6,(a4) ; put MS in arg1
pea arg1
pea arg2
fLZX ; convert MS to extended in arg2

move.l d7,(a4) ; put SMS in arg1
pea arg1
pea arg2

fLZX ; convert SMS to extended in arg3
move.l #8388608,(a4) ; 2^23
pea arg1
pea arg4

fLZX ; convert 2^23 to extended in arg4
pea arg4
pea arg2

fdivx ; divide MS by 2^23 to move decimal point
pea arg4
pea arg3
fdivx ; divide SMS by 2**3 to move decimal point
pea two
pea arg3
fdivx ; SMS/2
pea arg2
pea arg2
fmulx ; MS*2
pea arg2
pea arg2
fsubx ; E in arg3
lea arg1,a0
move.l #4342944,(a0)
pes arg1
pes arg2
pes arg3
fl2x ; get 1000000*log base e of 10 in arg2
pes thousand
pes arg2
fdivx ; get three decimal places
pes thousand
pes arg2
fdivx ; now six decimal places
pes arg3
flnz ; take log base e of E
pes arg2
pes arg3
fmulx ; now Power = (10 * log base e of E)/(log base e of 10) in arg3
pes arg3
pes arg2
fx2x ; copy arg3 (Power) to arg2
pes arg2
flintx ; Truncate result
pes arg2
pes arg3
fsubx ; Now integer part in arg2, fractional part in arg3
pes thousand
pes arg3
fmulx ; get three decimal places
pes thousand
pes arg3
fmulx ; now six decimal places
pes arg2
pes arg1
fx2l ; convert decimal part to long integer
lea PDecimal,a0
move.l arg1,(a0)
pes arg3
pes arg1
fx2l ; convert fractional part to long integer
lea PRowt,a1
move.l arg1,(a1)
bl PResult
tat.l (a0)
beq PResult
neg.l (a1)
Print Result

PResult:

bsr  WDHA:TDraw

; Now put the WDHA in either hearing aid state or idle state
clr.w (sp) ; GetCtlValue returns a word
move.w AidControl,-(sp) ; the handle
_leGetCtlValue
lst.w (sp)+
beg
move.w#-1.d0 ; go to hearing aid mode
bra TCSetMode

TCSetMode:

move.w#-100.d0 ; go to idle mode

jc set

; GelCtlValue returns a word

move.w#197.d0 ; Get the handle
move.w#-1.d0 ; go to hearing aid mode
bra TCSetMode

TCAidOff:

move.w#-1100.d0 ; Go to idle mode

; Name: TCCvtBoxes
; Function: TCCvtBoxes actually does the work of filling the paramrec by:
; converting the text of the text boxes to their appropriate values, and by:
; calculating the sine and cosine factors from the specified frequency.
; Input: None
; Output: None

TCCvtBoxes:

movem.1 d0-d7/a0-a6,-(sp)
lea TextHandles,a4
move.w#ToneBursts,d0

TCCBLoop:

cmp.w #TextBoxes,d4
bge TCCBLoop

move.w#2,d5 ; *4 for longs
move.w(a4,d5.w),a0 ; get the text handle
_HLock : Lock the handle
move.w(a0),a2 ; Dereference the handle
move.w#0(a2),d6 ; get len
lea NumBuf,a6
move.b d6,(a6) ; store the length of the string
clr.w -(sp) ; make room for the result.
move.w(a0),(sp) ; get the text
_TEGetText
move.(sp)+,a3 ; get it in a3
move.w(a3),a0

_HLock : lock the handle
move.w(a0),a0 ; Dereference the handle, move src in a0
lea NumBufT,a1 ; Destination is NumBufT
move.w#0,d0 ; BlockMove expects length in d0
ext.w d0 ; expects a long
_BlockMove
lea NumBuf,a0
move.w#1,-(sp) ; StringToNum puts result in d0
lea offsets,a1
move.b (a1,d4.w),d1  ; get offset in paramrec of this entry  
  ext.w d1  ; make it a word.  
lea  paramrec,a0  ; get paramrec base address  
move.w0,(a0,d1.w)  ; store the value.  
move.l a3,a0  ; Unlock the text handle  
  _HUnlock  
move.l (a4,d5.w),a0  ; Unlock the TEHandle  
  _HUnlock  
add.w #1,d4  ; go to next box.  
  bra  TCCELoop  
  
TCCEDone:  
  ; Now compute the slope delta values which are 16384/sample count  
lea  paramrec,a4  
movel #16384,d0  
movew2(a4),d1  ; first do the rise time slope delta  
beq  RTSZero  
divu  d1,d0  
movew0,4(a4)  
bra  FTSDelta  
RTSZero:  
movew#7FFF,4(a4)  
FTSDelta:  
movel #16384,d0  
movew8(a4),d1  ; now do the fall time slope delta  
beq  RTSZero  
divu  d1,d0  
movew0,10(a4)  
bra  TCCalcTrig  
FTS0Zero:  
movew#7FFF,10(a4)  
TCCalcTrig:  
  ; Now send the parameters to the WDHA  
movewFreq,d0  
lea  arg1,a1  
movewd0,(a1)  
pas  arg1  
pas  arg3  ; arg3 will hold to frequency  
  : convert from integer to extended fp  
F2X  
  ; Compute burst amplitude  
movew Atten,d0  
bpl  AttenOK  
cir.w d0  
AttenOK:  
egw  d0  
lea  arg1,a0  ; store Atten from max output (dB) in arg1  
pas  arg1  ; dB gain  
pas  arg4  ; pdB gain  
F2X  ; convert from integer to extended fp  
pas  fp20dB  
  ; .25 * log base 10 of e = 8.65689638  
pas  arg4  ; pdB gain  
fdivx :db/fp20db (result in arg4)  
pas  arg4  ; base e exponential (dB ratio in arg4)  
  
  102
; scale it \(2E14\) to convert it to fixed point
fx2i
lea paramrec,a4
move.warg1,20(a4) ; store the burst factor

; compute sine and cosine factors
; first get \(2\pi f / fs\) in arg5
lea paramrec,a4
move.warg1,16(a4) ; store cosine factor

; Now do sine
lea paramrec,a4
move.warg2,16(a4) ; push sine factor
movem.1 (sp)+,d0-d7,a0-e6
rts

; WDHATC data declarations..........................
WDHATCPtr: DCL 0 ; WDHATC WindowPtr
AidControl: DCL 0 ; Hearing Aid On Control
IAControl: DCL 0 ; Input Attenuation Control
OAControl: DCL 0 ; Output Attenuation
FieldControl: DCL 0 ; Field Mike Control
ProbeControl: DCL 0 ; Probe Mike Control
StartControl: DCL 0 ; Start Button Control

; Which Text Edit Record is active?
WActive: dc.w -1 ; -1 means none are active

TextHandles:
dcb.1 TextBoxes,0

paramrec:
dc.w 1 ; tone burst count
dc.w 0 ; rise time sample count
dc.w 0 ; rise time slope delta
dc.w 16384 ; signal on sample count
dc.w 0 ; fall time sample count
dc.w 0 ; fall time slope delta
dc.w 16384 ; signal off sample count
dc.w 4224 ; gain/input select word
dc.w 0 ; cosine factor
dc.w 0 ; sine factor
dc.w 32000 ; burst amplitude
dc.w 512 ; probe sample count [currently a constant]
dc.w 32 ; probe sample multiplier [currently a constant]

; The following are not really a part of the paramrec, but currently must
; follow it for the routine TCCvIBoxes to work properly
Freq: dc.w 0
Att.: dc.w 0

; Power
PDecimal: dc.l 0
PFract: dc.l 0

offsets:
dc.b 0 ; tone burst count is first entry
dc.b 2 ; rise is second
dc.b 6 ; on count is fourth
dc.b 8 ; fall count is next
dc.b 12 ; off count is seventh
dc.b 26 ; frequency is 14th (not really a parameter)
dc.b 28 ; atten is 15th (not really a parameter)

TextRects:
dc.w TCCInitY+ToneBursts*20
dc.w TCCInitX-88
dc.w TCCInitY+ToneBursts*20+20
dc.w TCCInitX-20

dc.w TCCInitY+RiseCount*20
dc.w TCCInitX-88
dc.w TCCInitY+RiseCount*20+20
dc.w TCCInitX-20

dc.w TCCInitY+OnCount*20
dc.w TCCInitX-88
dc.w TCCInitY+OnCount*20+20
dc.w TCCInitX-20

dc.w TCCInitY+FallCount*20
dc.w TCCInitX-88
dc.w TCCInitY+FallCount*20+20
dc.w TCCInitX-20

dc.w TCCInitY+OffCount*20
dc.w TCCInitX-88
dc.w TCCInitY+OffCount*20+20
dc.w TCCInitX-20

dc.w TCCInitY+Frequency*20
dc.w TCCInitX-88
dc.w TCCInitY+Frequency*20+20
dc.w TCCInitX-20

dc.w TCCInitY+Attenuate*20
dc.w TCCInitX-88
dc.w TCCInitY+Attenuate*20+20
dc.w TCCInitX-20

WDMATCHBounds: ; Bounding rect for window
DC.W TCCinitY
DC.W TCCinitX
DC.W TCCinitY+200
DC.W TCCRight

ERect: ; Bounding rectangle for part to erase
DC.W TCCinitY-8
DC.W 0
DC.W TCCinitY+7*TCCHeight
DC.W TCCinitX

TRect:
DCL 0
DCL 0 ;For calculating various rectangles.

TPoint: DCL 0 ;For calculating mouse change.

WhichControl: DCL 0 ; A control handle, for temporary storage.

NumBuf: DCB 0 ; Buffer for number conversion (length here)
NumBufT: DCBB 79,0 ; Text here

KeyBuf: DCBB 80,0
| arg1  | dcb.w 8,0 | integer buffer |
| arg2  | dcb.w 8,0 | extended floating point buffer |
| arg3  | dcb.w 8,0 | extended floating point buffer |
| arg4  | dcb.w 8,0 | extended floating point buffer |
| arg5  | dcb.w 8,0 | extended floating point buffer |
| cosreg| dcb.w 8,0 | room for cosine factor |
| sinreg| dcb.w 8,0 | room for sine factor |
| xacc  | dcb.w 8,0 | extended accumulator |
| txreg  | dcb.w 8,0 | temporary extended register |
| pi    | d0 $4000$, $sc90a$, $s5604$, $s1893$, $s74bc$ |
| twopl | d0 $4001$, $sc90a$, $s5604$, $s1893$, $s74bc$ |
| zero  | d0 $0000$, $0000$, $0000$, $0000$, $0000$ |
| one   | d0 $31ff$, $8000$, $0000$, $0000$, $0000$ |
| fp1095| d0 $31ff$, $0999$, $9999$, $9999$, $999a$ |
| two   | d0 $4000$, $8000$, $0000$, $0000$, $0000$ |
| twox14| d0 $4004$, $8000$, $0000$, $0000$, $0000$ |
| twox15| d0 $400e$, $8000$, $0000$, $0000$, $0000$ |
| twox16| d0 $400f$, $8000$, $0000$, $0000$, $0000$ |
| ten   | d0 $4002$, $a000$, $0000$, $0000$, $0000$ |
| hundred| d0 $4005$, $c800$, $0000$, $0000$, $0000$ |
| thousand| d0 $4008$, $fa00$, $0000$, $0000$, $0000$ |
| fp12500| d0 $400c$, $6e350$, $0000$, $0000$, $0000$ |
| fp12277| d0 $400c$, $6e3d4$, $0000$, $0000$, $0000$ |
| fp20dB| d0 $4002$, $8af9$, $dc22$, $dc0e5$, $8042$ |
This file must be included if your program uses the WDHA Test/Calibrate window.

XREF WDHATCOpen
XREF WDHATCClose
XREF WDHATCShow
XREF WDHATCHide
XREF WDHATCDraw
XREF WDHATCControl
XREF WDHATCIdle
XREF WDHATCKey
XREF WDHATCIS
XREF WDHATCDoTest
This file contains two routines which read text files containing numeric expressions, and download the numbers to the digital hearing aid. The routine WDHAFCSet is used in the Aid13 program to download filter tap coefficients to the hearing aid. The routine WDHASetFileParams is used to download parameters for the SS15 spectral shaping program.

The text files accessed by these routines must contain integer numbers separated by any character which is nonnumeric and not '-' (generally spaces, tabs, or carriage returns). The text files accessed by WDHAFCSet can also contain simple numeric expressions of the form \( A/B \), where \( A \) and \( B \) are integers.

Include MacTraps.D
Include ToolEq.x.D
Include SysEq.x.D
Include QuickEq.x.D
Include FSequ.D
Include MDS2:WDHADisk.hdr
Include MDS2:WDHASCI.hdr

XDEF WDHAFCSet
XDEF WDHASetFileParams

; Constants for division
NoDiv EQU 0 ; Haven't seen a '/'
ReadOne EQU 1 ; Read first operand
DoDiv EQU 2 ; Read second operand, so don't division.

; Name: WDHAFCSet
; Function: This routine uses the SFGetFile dialog to get the name of the file from the user, then opens the file, converts it's contents from text form to binary integer form, then downloads it to the hearing aid.
; Input: None
; Output: None
WDHAFCSet:

movem.l d0-d7/a0-a6,-(sp) ;

Do SFGetFile

mov.l #$00480048,-(sp) ; where

lea "Which Filter Coefficient File?" ; prompt

move.l #0,-(sp) ; fileFilter procedure

move.w #1,-(sp) ; display all types of files

lea FTypes ; typeList

move.l #0,-(sp) ; digHook

lea Reply ; SFReply

move.w #2,-(sp) ; trap to SFGetFile

Pack3

; Did they choose a file?

lea good,a3

tst.w (a3)

beq DoneFCSet

; Yes, open it

lea fName,a1 ; file name pointer

bsr DiskOpen

tst.w d1 ; test ioResult

bne DoneFCSet
: Now d2 has ioRefNum
move.w #1,d1 ; read one sector
lea myBuffer,a1
bsr DiskRead
bsr DiskClose
: Now convert text buffer to words
move.w #64,d3 ; d3 will be a counter
move.w #NoDiv,d6 ; d6 tells if we should divide or not
lea myBuffer,a1
lea numRec,a2
FCLoop:
lea numBuffer,a0
: Convert from text buffer to a string
ctr.w d4 ; count length of string
FCSLoop:
move.b (a1)+,d5
 cmp.b #1',d5
beq FCSNotDiv
move.w #ReadOne,d6
bra FCSDone
FCSNotDiv:
move.b #1',d5
beq FCSGo
move.w #0',d5
bne FCSDone
move.b #9',d5
bne FCSDone
FCSGo:
add.w #1,d4
move.b d5,(a0)+
bra FCLoop
FCSDone:
lea numString,a0
move.b d4,(a0)
move.w #1,-(SP) ; StringToNum - copy numString to word in d0
 cmp.w #NoDiv,d6 ; Are we dividing?
bne FCSDone1
 cmp.w #ReadOne,d6 ; Have we read one?
bne FCSDone2
add.w #1,d3 ; This one won't really count
move.w #DoDiv,d6 ; Next time we'll divide
bra FCSDone2
FCSDone1:
 cmp.w #DoDiv,d6 ; Should be dividing if we reach here
bne FCSDone2
move.w d0,d1 ; get the divisor in d1
lea -(a2),a2 ; back up the pointer to the first operand
move.w (a2),d0 ; get the first operand
ext.w d0 ; extend dest of divs to long
divs d1,d0
move.w #NoDiv,d6 ; finished this divide
bra FCSDone2
FCSDone2:
move wd0,(a2)+  ; store result
sub w #1,d3
bne FCLoop

; Send the coefficients to the WDHA
lea numRec,a0
bsr SetCoefficients

DoneFCSet:
movem.1 (sp)+,d0-d7/a0-a8
rts

; Name: WDHASetFileParams
; Function: This routine uses the WDHAGetFile dialog to get the file name
; from the user, then opens the file, converts it's contents from text form
; to binary integer form, then downloads it to the hearing aid.
; Input: None
; Output: None
WDHASetFileParams:

movem.1 d0-d7/a0-a8,-(sp)

; Do SFGetFile
movem.1 #$00480048,-(sp) ; where
lea "Which Set Params File?" ; prompt
move.1 #0,-(sp) ; FileFilter procedure
move.w #1,-(sp) ; display all types of files
lea FTypes ; typeList
deal.1 #0,-(sp) ; digFileProc
lea Reply ; SFRReply
move.w #2,-(sp) ; trap to SFGetFile

Pack3;
; Did they choose a file?
lea good,a3
stl.w (a3)
beq DoneFileSet

Yes, open it.
lea fName,a1 ; file name pointer
bsr DiskOpen
tst.w d1 ; test ioResult
bne DoneFileSet

Now d2 has ioRefNum
move.w #3,d1 ; read three sectors
lea myBuffer,a1
bsr DiskRead
bsr DiskClose

Now convert text buffer to words
move.w #320,d3 ; d3 will be a counter
lea myBuffer,a1
lea numRec,a2

FileOuterLoop:
lea numBuffer,a0

; Convert from text buffer to a string
c1r.w d4 ; count length of string

FileLoop:
move.b (a1)+,d5
cmp.b #",d5
beq FileGo
cmp.b #0',d5
bhi FileDone
cmp.b #9',d5
bhi FileDone

FileGo:
add.w #1,d4
move.b d5,(a0)+
bra FileLoop

FileDone:
lea numString,a0
move.b d4,(a0)
move.w #1,-(SP)
lea Pack7
move.wd0,(a2)+
sub.w #1,d3

Pack7

one FileOuterLoop

; Send the coefficients to the WDHA
lea numRec,a0
bsr SetFileParams

DoneFileSet:
movem.i (sp)+,d0-d7/a0-a6

rts

Reply:
good: dc.w 0
copy: dc.w 0
fType: dc.w 0
vRefNum: dc.w 0
version: dc.w 0
fName: dcb.b 64,0

fTypes: dc.l 'TEXT'

numString: dcb.b 0 ; length
numBuffer: dcb.b 63,0 ; text

numRec: dcb.w 320,0
myBuffer: dcb.b 1538,0
: WDHAFChdr
: This file must be included if your program uses the
: Set Filter Coefficients function.
XREF  WDHAFCSet
XREF  WDHASetFileParams
This file contains routines for sending records back and forth between the Mac and the WDHA via the SCSI bus interface.

Include MacTraps.D
Include SysEqxD.D
Include ToolEqxD.D
Include MDS2:WDHA.hdr

XDEF SetParam
XDEF SetCoefficients
XDEF SetFileParams
XDEF wdbTest
XDEF SCStInterrogate
XDEF SCStWr
XDEF SCStRd
XDEF SCStBTst

;scsi bus bit assignments
abs equ 1 ;assert data bus
dbs equ 0 ;deassert data bus
ack equ 0 ;assert acknowledge line
dck equ 16 ;deassert acknowledge line
atn equ 0 ;assert attention line
dtn equ 2 ;deassert attention line

;Set WDHA parameters subroutine
;calling protocol
lea paramrec,a0 ;set pointer to set parameter record
jsr SetParam
SetParam:

movem.l a0-a6/d0-d7-(sp) ;save registers
crl.w -(sp)
bsr SCStInterrogate
move.w (sp)+,d0
beq @4
cmp.w #-100,d0 ;SS15ID
beq @4
move.l #8-1,d1 ;set loop counter
move.w #2,d0 ;get -2 mode code (set aid parameters)
jsr ScsiWr ;send mode code to WDHA
@1 jsr ScsiBTst ;test for WDHA
beq @1 ;ready
@2 move.w (a0)+,d0 ;get parameter
jsr ScsiWr ;send parameter to WDHA
@3 jsr ScsiBTst ;test for WDHA
beq @3 ;ready
dbra d1,@2 ;check end of loop
move.w (a0)+,d0 ;get last parameter
jsr ScsiWr ;send last parameter to WDHA
@4 movem.l (sp)+,a0-a6/d0-d7 ;restore registers
rts
Set WDHA filter coefficients subroutine:

Calling protocol:

```
lea correc,aO ; set pointer to array of coefficients
jsr SetCoefficients
```

SetCoefficients:

```
movem.l a0-a6/d0-d7,-(sp) ; save registers
move.w #-4,d0 ; get -4 mode code (set aid coefficients)
jsr scsiwr ; send mode code to WDHA
```

@1
```
jsr SciBTst ; test for WDHA
beq @1 ; ready
```

@2
```
movem.l #63,d1 ; set loop counter
move.w(aO)+,d0 ; get parameter
jsr scsiwr ; send parameter to WDHA
```

@3
```
jsr SciBTst ; test for WDHA
beq @3 ; ready
```

```
sub.w #1,d1 ; check end of loop
```

```
move.w(a0)+,d0 ; get last parameter
jsr scsiwr ; send last parameter to WDHA
movem.l (sp)+,a0-a6/d0-d7 ; restore registers
```

WDTA test subroutine:

Calling protocol:

```
lea paramrec,aO ; set pointer to set parameter record
jsr wdhtest
```

Upon exit:
```
d6 has the mean sum
```
; d7 has the square mean sum
wdhatest:

; save registers
movem.1 a0-a6/d0-d5,-(sp)
move.ws-3,d0 ; get -3 mode code (test/calibrate)
jsr scsiwr ; send mode code to WDHA

@1 jsr ScsiBTst ; test for WDHA
beq @1 ; ready
move.l #13,d1 ; set loop counter (do all but last)

@2 move.w(a0)+,d0 ; get parameter
jsr scsiwr ; send parameter to WDHA
subq.b #1,d1 ; check end of loop

; read probe sample
@4 jsr ScsiBTst
beq @4

; read mean sum
clr.l d0
jsr scsiwr ; write dummy to wdha
jsr scsird ; read high 16 bits
move.w d0,d6 ; store in d6
swap d6 ; get it in high word

clr.l d0
jsr scsiwr ; write dummy to wdha
jsr scsird ; read low 9 bits
move.w d0,d8 ; store in d8

asl.w #7,d8 ; shift it left to the most sig word.

asn.l #7,d6 ; shift the whole thing right.

; read the mean square sum
clr.l d0
jsr scsiwr ; write dummy to wdha
jsr scsird ; read high 16 bits
move.w d0,d7 ; store in d7
swap d7 ; get it in most sig word.

clr.l d0
jsr scsiwr ; write dummy to wdha
jsr scsird ; read low 9 bits
move.w d0,d7 ; store in d7

asl.w #7,d7 ; shift it left to the most sig word.

asn.l #7,d7 ; shift the whole thing right.

movem.1 (sp)+,a0-a6/d0-d5 ; restore registers

; Name: SCSiWr
; Function: Send the 16 bit integer in d0 to the hearing aid via the SCSI bus.
; Input: d0 contains the word to write.
; Output: None

SCSiWr:

movem.1 d0-d3,-(sp)
move.l #abs=deck+dln,$580011 ; assert data bus
move.w #1,d2 ; set the
roxr.w #1,d2 ; extend bit
move.w #17-1,d2 ; set loop counter

@1: roxl.w #1,d0 ; move in next bit
move.w d0,d1 ; copy d0
and.w #1,d1 ;mask is bit
move.b d1,$580001 ;writs to output data bus
move.b #abs+ack+dtn,$580011 ;assert acknowledge (clock into wdha)
move.b #abs+dck+dtn,$580011 ;assert acknowledge (clock into wdha)
dbra d2,@1 ;loop counter
move.w$10000,d3 ;write delay
@2 dbra d3,!2
move.b abs+dck+dtn,$580011 ;deassert data bus and all
movem.l (SP),d0-d3
rts

; Name: SCSIrD
; Function: Read a word from the SCSI bus in register d0.
; Input: None
; Output: d0 contains the word read
SCSIrd: movem.l d1-d3,-(SP)
move #15-1,d2 ;set loop counter
move.b #abs+dck+dtn,$580011 ;deassert data bus and all
movem.w #250,d3 ;deassert-assert delay
movem.l (SP),d1-d3
rts

; Test SCSI read bit (Bit 1). Returns with d0 = 0 or 2
SCSlTest:
; If the mouse button is pressed then stop communication
movem.l a0-a1/d0-d2,-(SP) ; save registers
clr.w -(sp)
Button
tsl.w (sp)+
bne StopCom
movem.l (sp)+,a0-a1/d0-d2
move.b #abs+dck+dtn,$580011 ;deassert data bus and all
move.b $580000,d0 ;read SCSI bus
and.w #2,d0 ;mask position 1
rts

; If the button is pressed during communication we set the hearing aid
; to idle and return to the main loop. Note that extra parameters may
; be left on the stack from the routines which called SCSlTest.
StopCom:
move.w#-5,d0
bsr SCSlWr
bsr SCSIWr
movem.l (SP)+,a0-a1/d0-d2 ; Restore registers
clr.l (SP)+ ; Pop SCSlTest return address
bra EventLoop

; Name: SCSInterrogate
; Function: Interrogate the hearing aid to determine which program it is running, returning the program identifier code that the hearing aid sends back.
; If the hearing aid does not respond within a certain timeout period, the routine returns with zero as the result.
; Input: None
; Output: The program code (on the stack)

; **Note: The user should push a word for the result.

SCSInterrogate:

movem.l d0-d7/a0-a6-(sp) ;interrogate WDHA for program type
bsr SCSIWr
clr.w d0
move.w #20000,d7
@1 sub.w #1,d7
beq @2
jsr ScsBTst test for WDHA
beq @1 ;ready
@2 jsr scsrId ;read high 16 bits into d0
move.w d0,64(sp)
move.w #-1,d0 ;set hearing aid mode
bsr SCSIWr
movem.l (sp)+,d0-d7/a0-a6
rts
WDHASCSI.hdr

XREF SetParam
XREF SetCoefficients
XREF SetFileParams
XREF SCSIInterrogate
XREF wdhatest
XREF SCSiWr
XREF SCSiRd
XREF SCSiBTest

PROBE EQU 9
FIELD EQU 12
INPUT EQU 7
OUTPUT EQU 10
;WDHADisk.asm file

Include FSInclude
Include MacTrapsD ; Use System and Toolbox traps
Include ToolEquXD ; Use Toolbox equates
Include SysEquXD
Include QuickEquXD

XDEF DiskCreate
XDEF DiskRead
XDEF DiskWrite
XDEF DiskEject
XDEF DiskOpen
XDEF DiskClose
XDEF DiskSetFPos
XDEF DiskSetEOF
XDEF DiskSetFlnfo

ioNamePtr equ 18 ;not included in .d files
ioFVenNum equ 26 ;not included in .d files
ioMisc equ ioRefNum+4 ;not included in .d files

DiskRead:
:assumes d2 contains ioRefNum
:assumes d1 contains number of 512 byte sectors to read
:assumes a1 points to the buffer to fill
:returns with a0 pointing to parameter block on stack
:and with ioResult in d0
:the number of bytes actually read is returned in d3 (long)

moveq #ioVQEISize/2 - 1.d0
@1:  clr.w -(sp) ; make room on stack
      dbra d0,@1 ; for parameter block
      move.l sp,a0 ; set A0 for file manager call

move.w d2,ioRefNum(a0) ; to access parameters in block
mulu #512,d1 ; multiply number of sectors by 512
move.l d1,ioReqCount(a0) ; sectors required
divu #512,d1 ; restore d1
move.l a1,ioBuffer(a0)
_Read
move.l ioActCount(a0),d3
add #ioVQEISize,SP
rts

DiskWrite:
:assumes d2 contains ioRefNum
:assumes d1 contains number of 512 byte sectors to write
:assumes a1 points to the buffer to write
:returns with ioResult in d0
:and a0 pointing to parameter block on stack
moveq #ioVQESize/2 - 1,d0

@1: clr.w -(sp) ; make room on stack for move.l sp,a0 
    ; set A0 for file manager call move.w d2,loRefNum(a0) ; and to access parameters in block mulu #512,d1 ; sectors to write * 512 = bytes move.l d1,loReqCount(a0) ;blocks of 512 bytes required divu #512,d1 ;restore d1 move.l a1,ioBuffer(a0) ;_Write add #ioVQESize,sp rts

DiskSetFPos:
; assumes d2 contains ioRefNum 
; assumes d1 contains sector number to position at. 
; returns with ioResult in d0 
; and a0 pointing to parameter block on stack moveq #ioVQESize/2 - 1,d0

@1: clr.w -(sp) ; make room on stack for move.l sp,a0 
    ; set A0 for file manager call move.w d2,loRefNum(a0) ; and to access parameters in block move.w #1,loPosMode(a0) ;0 at current position ;1 relative to beginning of media ;3 relative to current position mulu #512,d1 move.l d1,loPosOffset(a0) ;blocks of 512 bytes required divu #512,d1 ;_SetFPos add #ioVQESize,sp rts

DiskClose:
; assumes d2 contains ioRefNum 
; returns with ioResult in d0 
; and a0 pointing to parameter block on stack moveq #ioVQESize/2 - 1,d0

@1: clr.w -(sp) ; make room on stack for move.l sp,a0 
    ; set A0 for file manager call move.w d2,loRefNum(a0) ; and to access parameter block close add #ioVQESize,sp rts

; d3 contains the drive number to eject

DiskEject:
moveq #ioVQESize/2, -1, d0

d@1:
c1r.w -(sp)
dbra d0, @1
move.l sp, a0
move.w #5, ioRefNum(a0)
move.w d3, ioDrvNum(a0)
move.w @ejectCode, csCode(a0)

_Eject
add #ioVQESize, SP
rts

DiskCreate:
:assumes a1 pointing to file name buffer
:returns with a0 pointing to parameter block on stack
:d3 contains the drive number to create the file on.

moveq #ioVQESize/2, -1, d0
@d1:
c1r.w -(sp)
dbra d0, @1
move.l sp, a0
move.l a1, ioNamePtr(a0)
move.b #0, ioFVersNum(a0)
move.w d3, ioVRefNum(a0)

_Create
add #ioVQESize, SP
rts

DiskOpen:
:assumes a1 pointing to file name buffer
:returns with a0 pointing to parameter block on stack
:ioRefNum in d2 and ioResult in d1
:upon return d3 contains the drive number the file was found on.

moveq #ioVQESize/2, -1, d0
@d1:
c1r.w -(sp)
dbra d0, @1
move.l sp, a0
move.l a1, ioNamePtr(a0)
move.b #0, ioFVersNum(a0)
move.w d3, ioRefNum(a0)
move.w #2, d2
beq DCopenGood
move.w @Open, d1

_Open
move.w #2, d3
move.ioRefNum(a0), d2
move.ioResult(a0), d1
beq DCopenGood
move.w #1, ioRefNum(a0)

Open
move.w #1, d3

121
move.w ioRefNum(a0),d2        ;save ioRefNum of file in d2
move.w ioResult(a0),d1        ;get io result

DOpenGood:
    add.l #ioVQEISize.SP       
    rts

DiskSetEOF:
    ;assumes d2 contains ioRefNum
    ;assumes d1 contains position to position at (a long)
    ;returns with ioResult in d0
    ;and a0 pointing to parameter block on stack
    moved #ioVQEISize/2 - 1,d0
    @1: clr.w -(sp)            ;make room on stack for
        dbra d0.@1           ;for parameter block
        move.l sp,a0        ;set A0 for file manager call
    move.w ioRefNum(a0)       ;and to access parameters in block
    move.w ioPosMode(a0)      ;0 at current position
    move.d1,ioMisc(a0)        ;1 relative to beginning of media
                             ;3 relative to current position
    _SetEOF
    move.w ioResult(a0),d0    ;get io result
    add.l #ioVQEISize.SP      
    rts

DiskSetFileInfo:
    ;assumes a1 pointing to file name buffer
    ;assumes d6 contains file creator
    ;assumes d7 contains file type
    ;d3 contains the drive number to create the file on.
    ;returns with a0 pointing to parameter block on stack
    movem.1 d0-d7/a6,-(sp)    ;and to access parameter block
    moved #ioVQEISize/2 - 1,d0
    @1: clr.w -(sp)           
        dbra d0.@1           ;set A0 for file manager call
        move.l sp,a0        ;and to access parameter block
        move.l a1,ioNamePtr(a0)  ;put name pointer in parameter block
        move.b #0,ioFVersNum(a0) ;version number, always use zero
        _GetFileInfo
        move.l a4,a0        ;drive #
        move.l [d7,32](a0)   
        move.l [d6,36](a0)   
        _SetFileInfo
    add.l #ioVQEISize.SP      
    movem.1 (sp)+,d0-d7/a6-a6  
    rts
; WDADisk.hdr
; This file must be included if your program uses the disk commands.
XREF DiskCreate
XREF DiskRead
XREF DiskWrite
XREF DiskEject
XREF DiskOpen
XREF DiskClose
XREF DiskSetFPos
XREF DiskSetEOF
XREF DiskSetFInfo
What is claimed is:
1. A hearing aid comprising:
a microphone for producing an input signal in response to sound; 
a plurality of channels connected to a common output, each channel comprising:
a filter with preset parameters for receiving the input signal and for producing a filtered signal;
a channel amplifier responsive to the filtered signal for producing a channel output signal;
a channel gain register for storing a gain value;
a channel preamplifier having a preset gain for amplifying the gain value to produce a gain signal;
wherein the channel amplifier is responsive to the channel preamplifier for varying the gain of the channel amplifier as a function of the gain signal; means for establishing a channel threshold level for the channel output signal; and means, responsive to the channel output signal and the channel threshold level, for increasing the gain value when the channel output signal falls below the channel threshold level and for decreasing the gain value when the channel output signal rises above the channel threshold level;

wherein the channel output signals are combined to produce an adaptively compressed and filtered output signal; and 
a transducer for producing sound as a function of the adaptively compressed and filtered output signal.
2. The hearing aid of claim 1 wherein the increasing and decreasing means in each of the channels comprises means for increasing the gain value in increments having a first preset magnitude and for decreasing the gain value in decrements having a second preset magnitude.
3. The hearing aid of claim 2 wherein the increasing and decreasing means in each of the channels further comprises:
a comparator for producing a control signal as a function of the level of the channel output signal being greater or less than the channel threshold level; and 
an adder responsive to the control signal for increasing the gain value by the first preset magnitude when the channel output signal falls below the channel threshold level and for decreasing the gain value by the second preset magnitude when the channel output signal rises above the channel threshold level.
4. The hearing aid of claim 1 wherein the filters in the channels have preset filter parameters for selectively altering the input signal over substantially all of the audible frequency range.
5. The hearing aid of claim 1 wherein each filter in the channels has preset filter parameters for selectively passing the input signal over a predetermined range of audible frequencies, each filter substantially attenuating any of the input signal not occurring in the predetermined range.
6. A hearing aid comprising:
a microphone for producing an input signal in response to sound;
a plurality of channels connected to a common output, each channel comprising:
a filter with preset parameters for receiving the input signal and for producing a filtered signal;
a channel amplifier responsive to the filtered signal for producing a channel output signal;
a channel gain register for storing a gain value;
a channel preamplifier having a preset gain for amplifying the gain value to produce a gain signal;
wherein the channel amplifier is responsive to the channel preamplifier for varying the gain of the channel amplifier as a function of the gain signal; means for establishing a channel threshold level for the channel output signal; and means, responsive to the channel output signal and the channel threshold level, for increasing the gain value when the channel output signal falls below the channel threshold level and for decreasing the gain value when the channel output signal rises above the channel threshold level;
a second channel amplifier responsive to the filtered signal for producing a second channel output signal; and 
means for programming the gain of the second channel amplifier as a function of the gain value for the respective channel;

wherein the second channel output signal is combined with the second channel output signals of the other channels for producing a programmably compressed and filtered output signal; and 
a transducer for producing sound as a function of the programmably compressed and filtered output signal.
7. The hearing aid of claim 6 wherein the programming means in each channel comprises means for varying the gain of the second channel amplifier as a function of a power of the gain value for the respective channel.
8. The hearing aid of claim 6 wherein the filters in the channels have preset filter parameters for selectively altering the input signal over substantially all of the audible frequency range.
9. The hearing aid of claim 6 wherein each filter in the channels has preset filter parameters for selectively passing the input signal over a predetermined range of audible frequencies, each filter substantially attenuating any of the input signal not occurring in the predetermined range.
10. A hearing aid comprising:
a microphone for producing an input signal in response to sound;
an amplifier for receiving the input signal and for producing an output signal;
means for establishing a threshold level for the output signal;
wherein the amplifier stores the gain setting as a first plurality of least significant bits and as a second plurality of most significant bits;
wherein the first preset magnitude comprises a number of bits less than or equal to a total number of bits comprising the least significant bits.
11. The hearing aid of claim 10 wherein the amplifier comprises a two stage amplifier, the first stage having a variable gain and the second stage having a predetermined gain.
12. The hearing aid of claim 10 further comprising means for producing a timing sequence wherein the gain register is enabled in response to the timing sequence for receiving the gain setting increase or decrease from the adder during a predetermined portion of the timing sequence.

13. The hearing aid of claim 10 wherein the adder further comprises a secondary register for storing a first and second preset magnitude and wherein the adder is responsive to the secondary register for increasing the gain setting in increments corresponding to the first preset magnitude and for decreasing the gain setting in decrements corresponding to the second preset magnitude.

14. The hearing aid of claim 10 further comprising means for clipping the output signal at a predetermined level and for producing an adaptively clipped compressed output signal.

15. The hearing aid of claim 10 further comprising means for clipping the output signal at a predetermined level and for producing an adaptively clipped compressed output signal.

16. The hearing aid of claim 10 further comprising a register for storing the first and second preset magnitudes, the register having six bits of memory for storing the first preset magnitude and six bits of memory for storing the second preset magnitude.

17. The hearing aid of claim 10 further comprising a register for storing the first and second preset magnitudes; wherein the register stores both said magnitudes in logarithmic form.

18. The hearing aid of claim 17 further comprising a limiter for limiting the output signal; wherein the limiter clips a constant percentage of the output signal.

19. A hearing aid comprising:
   a microphone for producing an input signal in response to sound;
   an amplifier for receiving the input signal and for producing an output signal;
   means for establishing a threshold level for the output signal;
   a comparator for producing a control signal as a function of the level of the output signal being greater or less than the threshold level;
   a gain register for storing a gain setting;
   an adder responsive to the control signal for increasing the gain value in increments having a first preset magnitude and for decreasing the gain value in decrements having a second preset magnitude;
   means for programming the gain of the amplifier as a function of the gain setting;
   a second amplifier responsive to the input signal for producing a second output signal;
   means for programming the gain of the second amplifier as a function of the gain setting in the gain register; and
   a transducer for producing sound as a function of the second output signal.

20. The hearing aid of claim 19 wherein the programming means comprises means for varying the gain of the second amplifier as a function of a power of the gain setting in the gain register.

21. A hearing aid comprising a plurality of channels connected to a common output, each channel comprising:
   a filter with preset parameters for receiving an input signal in the audible frequency range for producing a filtered signal;
   a channel amplifier responsive to the filtered signal for producing a channel output signal;
   a channel gain register for storing a gain value;
   a channel preamplifier having a preset gain for amplifying the gain value to produce a gain signal;
   wherein the channel amplifier is responsive to the channel preamplifier for varying the gain of the channel amplifier as a function of the gain signal;
   means for establishing a channel threshold level for the channel output signal and for decreasing the gain value when the channel output signal falls below the channel threshold level and for decreasing the gain value when the channel output signal rises above the channel threshold level; wherein the channel output signals are combined to produce an adaptively compressed and filtered output signal.

22. The hearing aid of claim 21 wherein the increasing and decreasing means in each of the channels comprises means for increasing the gain value in increments having a first preset magnitude and for decreasing the gain value in decrements having a second preset magnitude.

23. The hearing aid of claim 22 wherein the increasing and decreasing means in each of the channels further comprises:
   a comparator for producing a control signal as a function of the level of the channel output signal being greater or less than the channel threshold level; and
   an adder responsive to the control signal for increasing the gain value by the first preset magnitude when the channel output signal falls below the channel threshold level and for decreasing the gain value by the second preset magnitude when the channel output signal rises above the channel threshold level.

24. The hearing aid of claim 23 wherein the adder in a particular one of the channels further comprises a secondary register for storing the first and second preset magnitudes for the particular channel; and wherein the particular adder is responsive to the secondary register for increasing and decreasing the gain value in the particular channel gain register by said first and second magnitudes.

25. The hearing aid of claim 21 further comprising means for producing a timing sequence; wherein the channel gain register in at least one of the channels is enabled in response to the timing sequence for receiving the gain value from the respective adder during a predetermined portion of the timing sequence.

26. The hearing aid of claim 21 wherein each channel further comprises means for clipping the channel output signal at a predetermined level for producing an adaptively clipped and compressed channel output signal.

27. The hearing aid of claim 21 wherein the filters in the channels have preset filter parameters for selectively altering the input signal over substantially all of the audible frequency range.

28. The hearing aid of claim 21 wherein each filter in the channels has preset filter parameters for selectively passing the input signal over a predetermined range of audible frequencies, each filter substantially attenuating any of the input signal not occurring in the predetermined range.

29. The hearing aid of claim 21 wherein the filters in each of the channels comprise finite impulse response filters.

30. The hearing aid of claim 21 wherein each channel further comprises:
A hearing aid comprising:

- means responsive to the detecting means for producing a log value representative of the detected characteristic; and
- a memory for storing the look-up table comprising a preselected table of log values and related filter parameters and gain values,

said memory being responsive to the log value producing means for selecting a filter parameter and a gain value from the look-up table as a function of the produced log value, said variable filter being responsive to the memory for varying the parameters of the variable filter as a function of the selected filter parameter, and said amplifier being responsive to the memory for varying the gain of the amplifier as a function of the selected gain value.

36. A hearing aid comprising:

- a microphone for producing an input signal in response to sound;
- a plurality of channels connected to a common output, each channel comprising a filter with preset parameters for receiving the input signal and for producing a filtered signal and an amplifier responsive to the filtered signal for producing a channel output signal;
- a second filter with preset parameters responsive to the input signal for producing a characteristic signal;
- a detector responsive to the characteristic signal for producing a control signal, the detector including means for programming the time constant of the detector;
- means responsive to the detector for producing a log value representative of the control signal; and
- a memory for storing a preselected table of log values and gain values; wherein the memory is responsive to the log value producing means for selecting a gain value from the preselected table for each of the amplifiers in the channels as a function of the produced log value, and wherein each of the amplifiers in the channels is responsive to the memory for separately varying the gain of the respective amplifier as a function of the respective selected gain value; and
- a transducer for producing sound as a function of the combined channel output signals;

wherein said hearing aid does not include the use of a microprocessor.

37. The hearing aid of claim 36 wherein the filters in the channels have preset filter parameters for selectively altering the input signal over substantially all of the audible frequency range.

38. The hearing aid of claim 36 wherein each filter in the channels has preset filter parameters for selectively passing the input signal over a predetermined range of audible frequencies, each filter substantially attenuating any of the input signal not occurring in the predetermined range.

39. The hearing aid of claim 36 wherein the filters in each of the channels comprise finite impulse response filters, and wherein the second filter comprises a finite impulse response filter.

40. The hearing aid of claim 36 wherein the second filter is constituted by one of the filters in one of the channels.

41. A hearing aid comprising:

- a plurality of channels connected to a common output, each channel comprising:
  - a filter with preset parameters for receiving an input signal in the audible frequency range and for producing a filtered signal;
- a means for programming the gain of the second channel amplifier as a function of the selected filter parameter, and said amplifier being responsive to the memory for varying the gain of the amplifier as a function of the selected gain value.

30. The hearing aid of claim 29 wherein the programming means in each channel comprises means for varying the gain of the second channel amplifier as a function of the stored power value.

31. The hearing aid of claim 30 wherein the programming means in each channel comprises means for varying the gain of the second channel amplifier as a function of the power of the stored power value.

32. The hearing aid of claim 31 wherein the programming means in each channel further comprises means for storing a power value and wherein the programming means varies the gain of the second channel amplifier as a function of the value derived by raising the gain value for the respective channel to the power of the stored power value.

33. The circuit of claim 32 wherein the first and second channel amplifiers of each channel each comprise a two stage amplifier, the first stage having a variable gain and the second stage having a preset gain.

34. A hearing aid for use by a person having a hearing impairment spanning a predetermined frequency range, the hearing aid comprising:

- a microphone for producing an input signal in response to sound;
- only one broadband filtering channel spanning the predetermined frequency range of the hearing impairment, said channel comprising:
  - a variable filter with separately variable filter parameters for receiving the input signal and for producing an adaptively filtered signal; and an amplifier for receiving the adaptively filtered signal and for producing an amplified adaptively filtered output signal;
  - wherein said broadband filtering channel has a bandwidth corresponding to the predetermined frequency range of the hearing impairment;
- a preset filter with preset parameters responsive to the input signal for producing a characteristic signal;
- a detector responsive to the characteristic signal for producing a control signal, the detector including means for programming the time constant of the detector;
- means responsive to the detector for producing a log value representative of the control signal;
- a memory for storing a preselected table of log values and gain values; wherein the memory is responsive to the log value producing means for selecting a gain value from the preselected table for each of the amplifiers in the channels as a function of the produced log value, and wherein each of the amplifiers in the channels is responsive to the memory for separately varying the gain of the respective amplifier as a function of the respective selected gain value; and
- a transducer for producing sound as a function of the combined channel output signals;

wherein said hearing aid does not include the use of a microprocessor.
a channel amplifier responsive to the filtered signal for producing a channel output signal;
means for establishing a channel threshold level for the channel output signal;
a comparator for producing a control signal as a function of the level of the channel output signal being greater or less than the channel threshold level;
a channel gain register for storing a gain setting;
an adder responsive to the control signal for increasing the gain setting by a first preset magnitude when the channel output signal falls below the channel threshold level and for decreasing the gain setting by a second preset magnitude when the channel output signal rises above the channel threshold level; and
a second channel gain register for storing a predetermined channel gain value to define an operating range for the channel as a function of a signal level of the input signal;
wherein the channel amplifier is responsive to the gain register and to the second channel gain register for vary the gain of the channel amplifier as a function of the gain setting and the predetermined channel gain value; and
wherein the channel output signals are combined to produce an adaptively compressed and filtered output signal.
42. The hearing aid of claim 41 wherein the channel amplifiers each comprise a two stage amplifier, wherein the first stage has a predetermined gain for defining an operating range for the respective channel and the second stage has a variable gain responsive to the first stage.
43. The hearing aid of claim 42 wherein the first stage of each of the two stage amplifiers further comprises means for sequentially modifying the gains of each of the respective second stages from first to last as a function of the level of the input signal.
44. The hearing aid of claim 41 wherein the filters in the channels have preset filter parameters for selectively altering the input signal over substantially all of the audible frequency range.
45. The hearing aid of claim 41 wherein each filter in the channels has preset filter parameters for selectively passing the input signal over a predetermined range of audible frequencies, each filter substantially attenuating any of the input signal not occurring in the predetermined range.
46. The hearing aid of claim 41 wherein the filters in each of the channels comprise finite impulse response filters.
47. The hearing aid of claim 41 wherein the first and second magnitudes in a particular one of the channels are different numerically from the first and second magnitudes in another one of the channels.
48. The hearing aid of claim 41 wherein the adder in a particular one of the channels further comprises a secondary register for storing the first and second preset magnitudes for the particular channel and wherein the particular adder is responsive to the secondary register for increasing and decreasing the gain value in the particular channel gain register by said first and second magnitudes.
49. The hearing aid of claim 41 further comprising means for producing a timing sequence; wherein the channel gain register in at least one of the channels is enabled in response to the timing sequence for receiving the gain setting from the respective adder during a predetermined portion of the timing sequence.
50. The hearing aid of claim 41 wherein each channel further comprises means for clipping the channel output signal at a respective predetermined level for producing an adaptively clipped and compressed output signal.

* * * * *
UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,706,352
DATED : January 6, 1998
INVENTOR(S) : A. Maynard Engebretson et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 201, claim 33, line 21, "The circuit of" should read ---The hearing aid of---.

Column 197, claim 4, line 47, "The hearing of" should read ---The hearing aid of---.

Column 197, claim 5, line 51, "The hearing of" should read ---The hearing aid of---.

Column 198, claim 8, line 28, "The hearing of" should read ---The hearing aid of---.

Column 198, claim 9, line 32, "The hearing of" should read ---The hearing aid of---.

Signed and Sealed this
Twenty-ninth Day of September, 1998

Attest:

BRUCE LEHMAN
Attesting Officer

BRUCE LEHMAN
Commissioner of Patents and Trademarks