Fifth Quarterly Progress Report

April 1 through June 30, 2003
NIH Project N01-DC-2-1002

Speech Processors for Auditory Prostheses

Prepared by

Reinhold Schatzer, Marian Zerbi, Xiaoan Sun,

Jeannie Cox, Robert Wolford, Dewey Lawson and Blake Wilson

Center for Auditory Prosthesis Research
Research Triangle Institute
Research Triangle Park, NC

CONTENTS

I. Introduction

The main objective of this project is to design, develop, and evaluate speech processors for implantable auditory prostheses. Ideally, such processors will represent the information content of speech in a way that can be perceived and utilized by implant patients. An additional objective is to record responses of the auditory nerve to a variety of electrical stimuli in studies with patients. Results from such recordings can provide important information on the physiological function of the nerve, on an electrode-by-electrode basis, and can be used to evaluate the ability of speech processing strategies to produce desired spatial or temporal patterns of neural activity.

Work and activities in this quarter included:

- Seven days of continuing studies with local subject ME-16, implanted bilaterally with Med-El Tempo+ devices.
- Attendance by Blake Wilson at the *9th Symposium on Cochlear Implants in Children*, Washington DC, April 24-26.
- A visit by Jim Patrick, Senior Vice President of Research and Applications, Cochlear Ltd., Lane Cove, NSW, Australia, April 30.
- Evaluation of new subject ME-22, implanted with bilateral Med-El Tempo+ devices, May 9, June 6, and June 20.
- A visit by consultant Mariangeli Zerbi to verify software for modulated pulse train interaural timing difference studies, May 17-19.
- Two weeks of studies with returning bilaterally implanted subject ME-18, May19 – May 30.
- A visit by consultant Enrique Lopez-Poveda from Salamanca, Spain, May 19-23 coinciding with the visit of subject ME-18
- Participation by Blake Wilson in the *VII International Conference on Cochlear Implants and Related Audiological Sciences* and the opening of the Center for Hearing and Speech, Warsaw - Kajetany, Poland, May 22-25.
- A return visit by subject ME-14, June 16-17. This subject previously participated in our studies of combined electric/acoustic hearing and now is bilaterally implanted with Med-El Tempo+ devices.
- Participation by Blake Wilson in the celebration conference *25 Years of Cochlear Implants in Vienna*, Vienna, Austria, June 19.
- A return visit by bilaterally implanted subject ME-12, June 25 and 26.

In addition to the above-mentioned activities, work continued on the analysis of previously collected data, further development of the new dual resonance nonlinear (DRNL) filter processing strategy, and development of new measures of sensitivities to interaural timing differences for studies with recipients of bilateral cochlear implants.

In this report we provide a description of the hardware and software tools that have recently been developed and implemented in our speech processor laboratory to facilitate a time-efficient and flexible evaluation of novel and more complex speech processing schemes. With these new system tools (referred

to as “streaming mode” in this report), pre-processed pulsatile stimulation sequences are downloaded and delivered to the subject’s implant(s). These stimulation sequences are generated off-line, according to a given processing strategy implemented in the MATLAB programming environment, by processing sentence or consonant tokens in digital audio file format and converting them into a stimulation pulse sequence.

Results from other studies and activities indicated above will be presented in future reports.

II. Recent enhancements of the speech laboratory system

The last several quarters saw a significant improvement of the capabilities of our speech laboratory system. Real-time CIS speech processor designs, including bilateral capabilities, have been finalized for Med-El C40/C40+ and Nucleus CI24M and CI22 implant systems. Psychophysical test procedures for the same bilateral implant systems have been extended. Most recently, the capability to stimulate implant subjects with pulsatile sequences processed off-line from digitally stored speech tokens has been added. This feature is now available with uni- or bilateral Med-El and Nucleus CI24M patients, as well as subjects with direct access to their intracochlear electrodes through a percutaneous connector. A description of this download-and-stimulate feature is presented in this report. For more information about the bilateral tools, please see Appendix 2.

Streaming mode

Introduction

Our typical approach to designing speech processors for speech reception measurements with cochlear implant subjects is to implement the processing algorithms in a real-time environment on a digital signal processor (DSP). The advantage of this approach is that any speech test signal available in either analog or digital format, including live voice sampled through a microphone, can be easily fed into the speech processor. This provides the possibility for the subject to listen to familiar sounds and voices, including his or her own voice, whenever being exposed to new processing strategies or variations thereof, therefore making it much easier to adjust to or pick out potential differences. Also, it is very simple to mix speech signals into different noise backgrounds and adjust the signal to noise ratio as desired.

However, a major drawback of a real-time approach for implementing speech processor designs is the generally complex and time-consuming implementation task itself. For the sake of optimal code execution time and memory management, our DSP programs are all written in assembly language. Also, processing algorithms must be carefully designed to run on a platform with fixed-point data representation (as is the case with the MOTOROLA DSP56301, which is the core of our lab system), in order to avoid data under- or overflows or undesired propagation of quantization errors due to finite word lengths.

A major point of investigation in the current contract period is to explore novel speech processing strategies for cochlear implants that more closely mimic the functions of the auditory periphery in normal hearing, like level-dependent tuning of the basilar membrane, or inner hair cell transduction processes. The likely complexity of such novel processing structures and the high cost in terms of development time to run them in real-time was the main motivation to look for alternative methods of implementing and evaluating new speech processor designs in an efficient, flexible, and fast way. This was achieved by integrating a new program mode (streaming mode) into the lab system, where pre-processed pulsatile stimulation sequences are downloaded and delivered to the subject's implant(s). These stimulation sequences are generated off-line in the MATLAB programming environment by processing speech tokens like sentences or consonants in digital audio file format and converting them into a stimulation pulse sequence, according to a given processing strategy also implemented in MATLAB. The powerful and highly flexible MATLAB environment supports a fast and relatively simple implementation of complex processing schemes.

Design specification

The requirements for the design and implementation of the streaming mode can be summarized as:

- Support for bilateral Med-El and Nucleus CI24M interfaces, as well as for the current source interface for subjects with percutaneous access to their implanted electrodes.

- MATLAB routines receive basic speech processor parameters directly from the same speech processor specification ("spec") files used by the monitor program during stimulus presentation or in the speech processor control mode (see Figure 20 in Appendix 2 for an example). These basic parameters are number of channels (left/right), stimulation rate per channel (left/right), stimulation order of channels (left/right), and synchronization settings. They define in which order and at what times the speech processor channel output waveforms are sampled into pulse amplitudes.

- Input of additional speech processor parameters (filter characteristics, maplaw, etc.) through the command line or function parameters in MATLAB.

- Generation of "normalized" pulse amplitudes when sampling channel output signals in MATLAB into an amplitude sequence (AMP) file, with 0 corresponding to the threshold (THR) on each active electrode, and 1 corresponding to the most comfortable loudness level (MCL).

- Optimization of AMP file format in terms of file size, by packing pulse amplitudes into a binary byte sequence. Amplitude values are stored as 12-bit unsigned integers, so that two pulse amplitudes fit into one 24-bit DRAM memory word. Given the native 24-bit signed fractional data format of the MOTOROLA DSP56301 (MOTOROLA, DSP 56300 Family Manual, Rev. 3.0, 11/2000, section 3.3.1), 12-bit amplitude values are left aligned to bit #22 (after the decimal point) in the DSP, which corresponds to a "type cast" into a signed positive 13-bit fractional number in the range $[0, 1 - 2^{-12}] = [0, 0.999755859375]$.

- Mapping of normalized pulse amplitudes to actual dynamic ranges of active electrodes is done in the DSP through the following linear transformation:

  $$amp_i = (MCL_i - THR_i) * \text{normalized } amp_i + THR_i, \qquad i\ldots\text{electrode index} \qquad (1)$$

  With the bilateral Med-El and CI24M interfaces, the stimulus pulse amplitudes are scaled by a linear channel specific volume and a master volume factor, both ranging from 0 to 100%, before being sent to the implant. All volume factors are independently adjustable for left and right sides. With the current source interface, only the channel specific volume is applied in the DSP, whereas the master volume is controlled through a potentiometer on the operator panel.

  The monitor program prepares channel data tables in the DSP (including THR/MCL and pulse duration values, stimulation order, and rate of channels) by scanning a speech processor spec file and downloading parameters to DSP memory. Basic stimulation parameters (number of channels, stimulation rate, stimulation order, and synchronization settings) must match between the AMP file and the spec file and must be checked by the monitor program for each AMP file downloaded and streamed to the subject's implant(s). The operator controls AMP file downloads and execution and stimulation volumes through the monitor program.

- Stimuli executed from a downloaded AMP file sequence are "embedded" in a background pattern of stimulation pulses at the specified rate and phase duration and at amplitudes

  $$THR_i * vol_i * (100 + \text{background balance})/100 \qquad vol_i\ldots\text{channel specific volume} \qquad (2)$$

i … electrode index

for the bilateral Med-El and CI24M interfaces, and

$THR_i * (100 + background\ balance)/100$ *i*…*electrode* index (3)

for the current source interface. In other words, for the time of the signal duration in the AMP file, stimulation pulse amplitudes are taken from the downloaded amplitude sequence, whereas outside of the stimulus, pulse amplitudes are constant and at the levels indicated above. The background balance can be set from –100 to +20 for adjusting volume-scaled threshold pulse amplitudes anywhere from –100% to +20%. In the case of the bilateral interfaces, a single background balance control applies to both left and right sides.

- Support for unilaterally as well as bilaterally implanted subjects, *i.e.* ability to generate unilateral and bilateral amplitude sequence files. Ability to play a unilateral AMP file sequence bilaterally by reproducing the stimulus pulse sequence on both sides (normalized amplitudes in the stimulus pulse sequence are scaled to the appropriate electrical dynamic ranges of each electrode on both ipsilateral and contralateral sides).

- Option to synchronize and specify a delay of relative pulse onset timing between the two sides for bilateral implants.

- Support for single-channel (monophonic) as well as dual-channel (stereophonic) wave file input signals.

Figure 1 shows the flow of events from generating pulse amplitude sequence files (AMP files) out of digital wave file recordings in MATLAB, through downloading them into the DSP, to transmitting them as a stream of stimulation pulses to the subject's implant(s) when executing the processed stimulus.

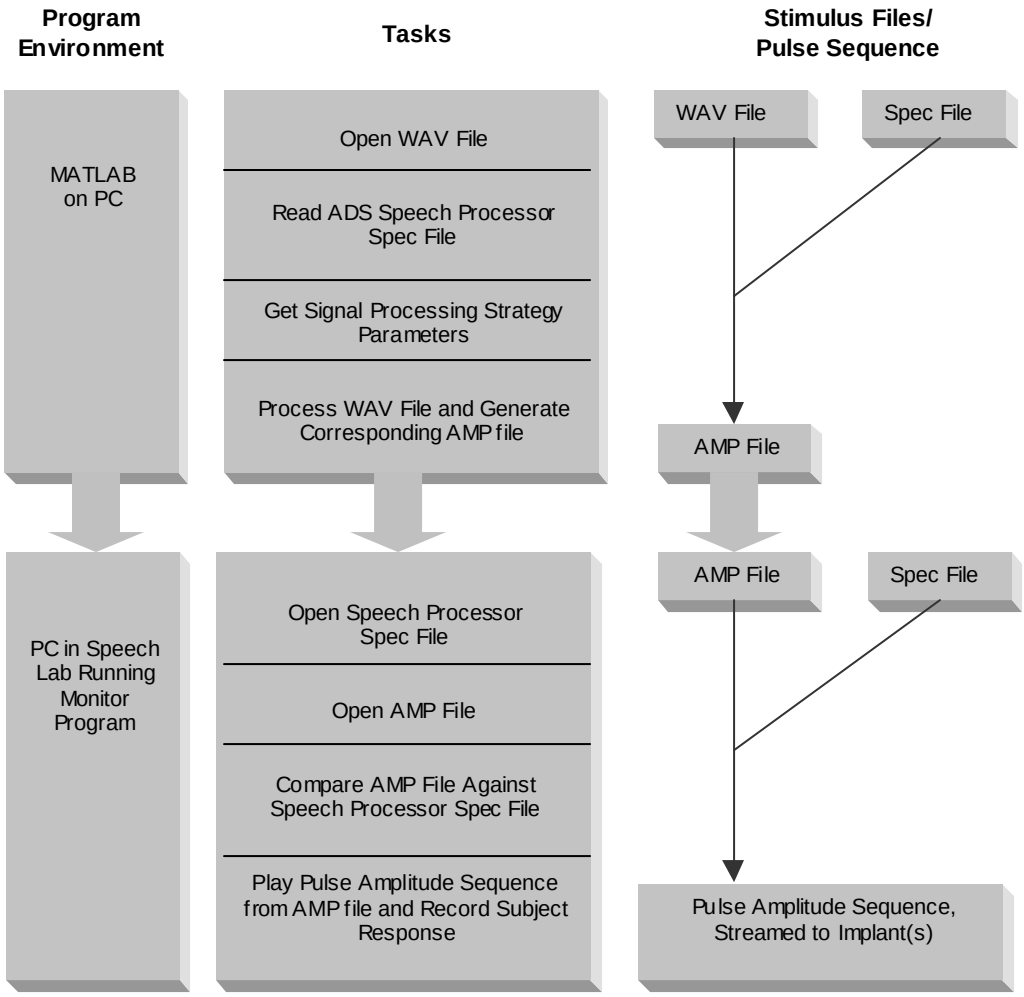


Figure 1: Paradigm of the streaming mode implementation and operation

DSP program

The basic DSP program structure of the streaming mode, both for the bilateral Med-El and CI24M interfaces and for the parallel current source interface, is quite similar to the real-time processor DSP program structure described in section "DSP programs for bilateral speech processors" of Appendix 2. The essential difference is that in the streaming mode, pulse amplitudes are not derived through sampling of the channel output envelopes that are processed from the ADC input signal in real-time, but rather are taken from the AMP file pulse amplitude sequence downloaded into DSP memory.

In the streaming mode, the monitor program runs through the following sequence of steps to set up the DSP memory and execute stimuli consisting of pulse sequences downloaded from pre-processed AMP files:

1. Download DSP program code over the on-chip emulator (OnCE) port of the ADS56301 lab processor module (see Figure 10 of Appendix 2). Start DSP program and command DSP into the streaming program mode.

2. Open speech processor spec file containing patient-specific THR/MCL values and stimulation parameters, such as active electrodes, phase duration, and pulse rate. Prepare tables (channel data tables) in DSP memory with all relevant parameters.

3. Download binary amplitude sequence(s) contained in AMP file(s) into DSP external DRAM memory. DRAM memory size is 1.5 Mbytes on the ADS56301.

4. Start background stimulation of volume scaled THR pulses on all active electrodes.

5. Execute stimulation sequences downloaded from AMP files in randomized order (for 16 or 24 medial consonant tests) or sequential order (sentence tests). Record results for consonant tests (sentences are scored manually). If necessary, download AMP file for each new stimulus after the execution of the previous one.

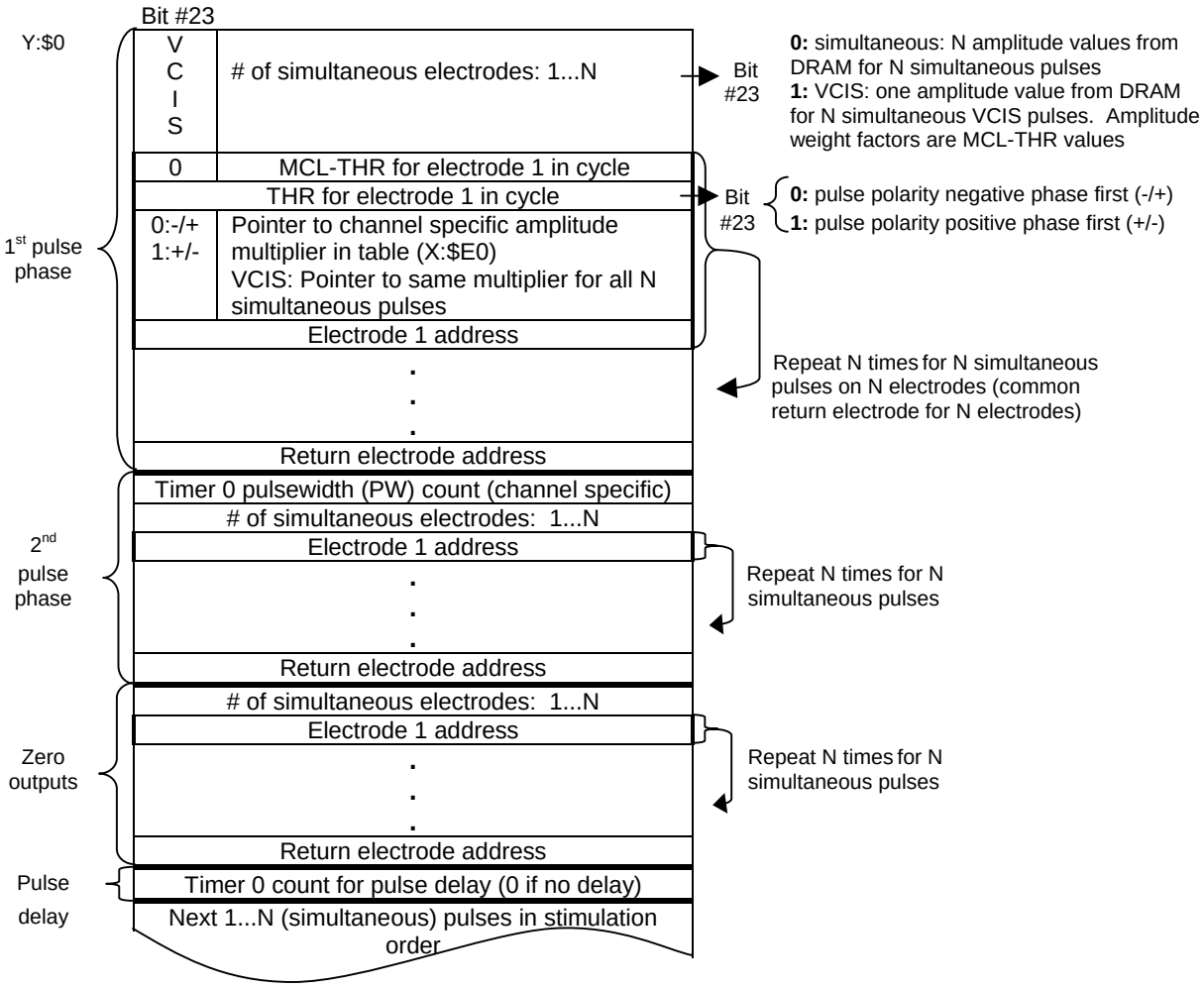
<u>DSP memory organization and pulse data transfer to implant interfaces</u>

During stimulation, the DSP retrieves biphasic pulse parameters such as electrode address, phase duration, and pulse gap from the channel data tables initialized by the monitor program and transfers them to the implant interface for execution. Pulse amplitudes are scaled to electrode THR and MCL levels according to equation (1) and scaled by a volume factor that is adjustable through the monitor program, before being sent to the interface. With the bilateral implant interfaces, the interface triggers an interrupt in the DSP in order to request new pulse parameters, for either the left or right side. The frequency of the interface requests is determined by the overall pulse rate on each side. With the current source interface, the pulse output timing is controlled by a timer in the DSP. This timer triggers the events when current pulse amplitudes (in 12-bit signed format) are being written to the interface.

<u>Channel data table format for parallel current source interface</u>

The streaming mode for the current source interface supports both interleaved and simultaneous pulse processors. Stimulation pulse data are arranged in two channel data tables in DSP memory: One for stimulus pulses at address range Y:$0 - Y:$3FF and one for background pulses in address range Y:$400 - Y:$7FF.

1. Tables for interleaved, simultaneous and virtual CIS processors



Channel specific volume table:

X:$E0	Volume factor channel 1
.	Volume factor channel 2
:	:
X:FF_{max}$	Volume factor channel (number of channels)

One use of simultaneous pulses in in virtual CIS (VCIS) strategies, where two or more adjacent electrodes are stimulated simultaneously to obtain a perceived pitch distinct from that of any electrode alone.

As an example of the relationship between the channel data table and the spec file, consider a VCIS processor in which channel 1 stimulates electrodes 1 and 2 simultaneously with respect to return electrode 8, with electrode 2 receiving a stimulus that is one half the amplitude delivered to electrode 1 and of opposite polarity. The corresponding spec file would reflect this situation in the first line of its >VCIS section:

```
>VCIS
8,1:100,2:-50
. . .
```

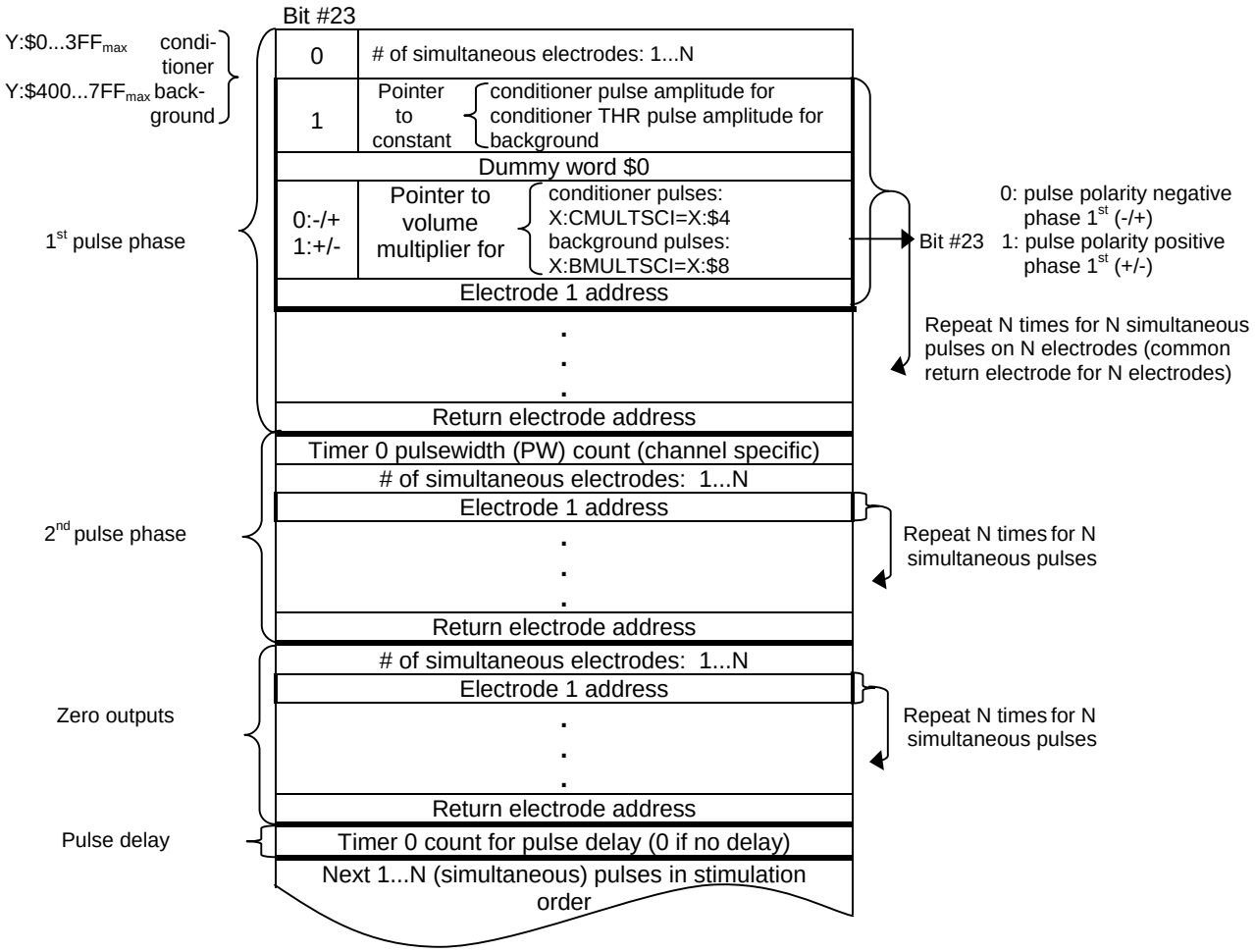
Later in the spec file, the first two lines of the psychophysics section would contain the THR and MCL values for electrodes 1 and 2, respectively:

```
>psycho
50 100
75 150
. . .
```

The corresponding channel data table would contain amplitude values calculated using the THR and MCL data for each electrode along with the appropriate VCIS weight multipliers. Its first nine entries would contain the following information (with appropriate settings of bit 23 where relevant):

Y:$0	2	[number of simultaneous electrodes for channel 1]
.	1.0 · (100-50) · DACunits/µA	[electrode 1 MCL-THR]
.	1.0 · (50) · DACunits/µA	[electrode 1 THR]
.	$E0	[pointer to VCIS channel 1 amplitude multiplier]
.	address for electrode 1	
.	-0.5 · (150-75) · DACunits/µA	[electrode 2 MCL-THR]
.	-0.5 · (75) · DACunits/µA	[electrode 2 THR]
.	$E0	[pointer to VCIS channel 1 amplitude multiplier]
.	address for electrode 2	
Y:$9	address for return electrode 8	

2. Tables for conditioner CIS and background stimulation



Conditioner amplitude table

X:$40	Conditioner amplitude 1
.	Conditioner amplitude 2
:	:
X:$5F$_{max}$	Conditioner amplitude (number of electrodes)

Background stimulation amplitude table (THR amplitudes)*

X:$80	THR amplitude(s) channel 1
.	THR amplitude(s) channel 2
:	:
X:DF_{max}$	THR amplitude(s) channel (number of channels)

*Note: For virtual CIS, background data for one virtual channel consist of N subsequent electrode amplitude values

Channel data table modulo values (values for M_x registers)

Stimulus channel data table modulo: X:TBL = X:$E
Background stimulus channel data table modulo: X:BKGTBLM = X:$B

3. NOTE: Channel data tables imply common return electrode for N simultaneous pulses on N electrodes

Channel data table format for bilateral Med-El and CI24M interfaces

1. Channel data table format for Med-El interface:

X:$100 LEFT	$MCL_1 – THR_1$	THR_1	$pVolTbl_1$	$MDIST_1$	DRN_1	for Med-El THR, MCL values are in least significant bits
	$MCL_2 – THR_2$	. . .	. . .	. . .	. . .	
	. . .					
X:$200 RIGHT						

2. Channel data table format for CI24M interface:

X:$100 LEFT	MCL_1 - THR_1	THR_1	$pVolTbl_1$	$PulseWord_1$	$PulseWord_2$	for CI24M THR, MCL values are in least significant bits
	MCL_2 - THR_2	. . .	. . .	. . .	. . .	
	. . .					
X:$200 RIGHT						

Channel data table modulo values: LEFT X:$58
RIGHT X:$59

3. Channel data tables for background stimulation with Med-El

LEFT X:$400	$0	THR_1	$pVolTbl_1$	$MDIST_1$	DRN_1
RIGHT X:$500	$0	THR_2	:	:	:

4. Channel data tables for background stimulation with CI24M

LEFT X:$400	$0	THR_1	$pVolTbl_1$	$PulseWord_1$	$PulseWord_2$
RIGHT X:$500	$0	THR_2	:	:	:

5. Background channel data table modulo: LEFT X:$56
RIGHT X:$57

Channel specific volume tables: X:$340 - X:$35F LEFT for Med-El and CI24M
(same as previous) X:$360 - X:$37F RIGHT

6. Overall volume: Stimulus: X:$32 LEFT, X:33 RIGHT
Background: X:$34 LEFT, X:35 RIGHT

Stimulus execution

Stimuli that are executed from amplitude sequences downloaded into DRAM memory are embedded in background stimulation running at the stimulus pulse rate. Background stimulation can be started and stopped at any time through dedicated byte commands given by the monitor program on the host PC and sent to the DSP over the serial COM port connection (see Figure 10 in Appendix 2). The embedding of stimuli is realized in the DSP by dynamically redirecting the channel data table read pointer(s) at the end of the background stimulation table(s) to the beginning of the stimulus channel data table, if a dedicated stimulus execute serial command had been received from the monitor. At the end of the stimulus, the channel data table pointer(s) automatically fall back from the stimulus table(s) into the background channel data table(s). For the bilateral Med-El and CI24M interfaces, a stimulus can be executed either on one side only or bilaterally.

Pulse amplitude sequences are stored in AMP files sequentially in the order in which they are sampled from the channel output signals. Thus, it is necessary that the following parameters match between the amplitude sequence in an AMP file and the carrier pulses generated in the implant(s), so that the waveforms modulated onto the carrier pulses are a true representation of the channel output signals sampled into the AMP file in MATLAB:

- Number of channels
- Stimulation order of channels
- Stimulation rate per channel (sampling rate of channel envelopes). For bilateral stimuli, the onset of the carrier pulses (samples) must be synchronized between the two sides.
- Synchronization setting and delay between sides (for bilateral stimuli)

Figure 2 shows oscilloscope traces of the pulse trains on channels 1 and 8 of an 8-channel standard CIS processor on a Med-El C40+ implant, right at the transition from background stimulation at –30% of THR to an overall stimulus volume of 100%. The stimulus amplitude sequence (file M0510.amp) has been processed from a wave file recording of the medial consonant ‘ASA’ (file M0510.wav). Figure 3 shows the same consonant’s channel output waveforms over the whole duration of the stimulus, embedded in background stimulation pulses, this time at THR + 0% background balance (and still 100% overall stimulus volume).

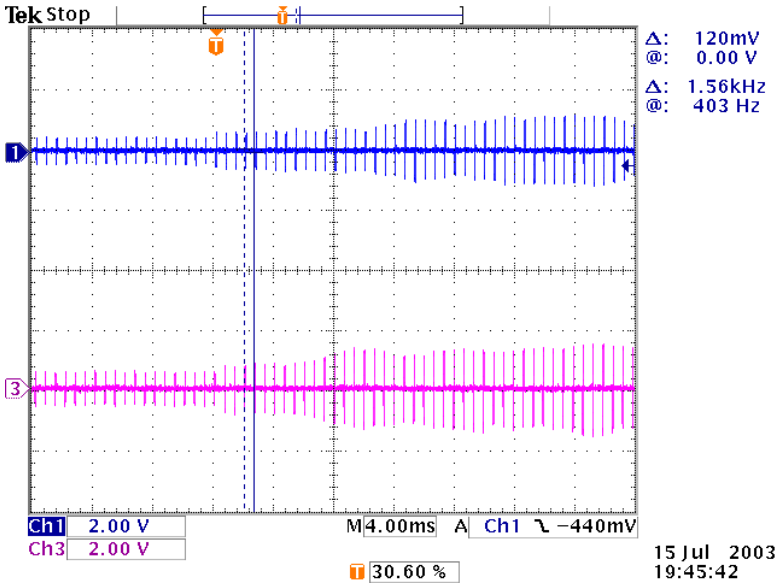


Figure 2: Oscilloscope traces of pulse trains on channels 1 (top) and 8 (bottom) of an 8-channel standard CIS processor with medial consonant ASA at the transition from background stimulation to stimulus onset

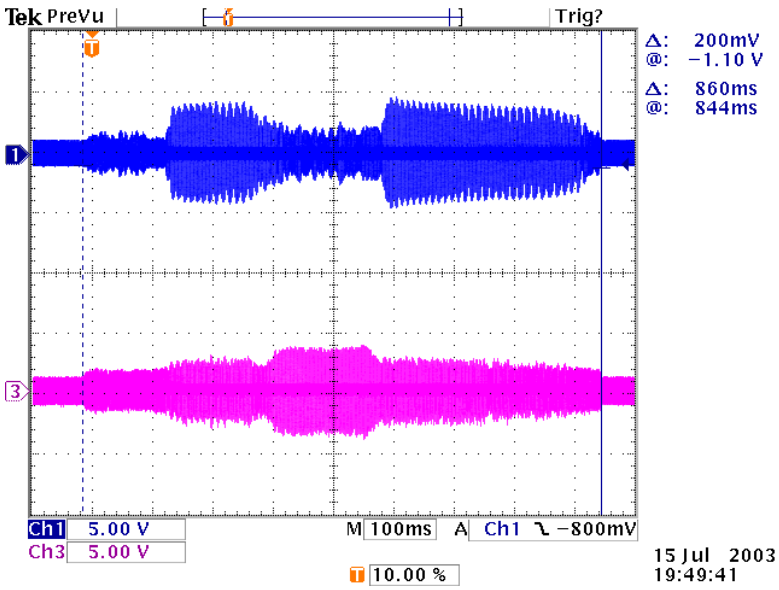


Figure 3: Oscilloscope traces of pulse train envelopes on channels 1 (top) and 8 (bottom) of an 8-channel standard CIS processor for a medial consonant ASA (M0510.amp). Overall stimulus duration is 860 ms, including 140 ms of leading "silence." See Figure 9 for the corresponding channel envelope signals as processed and displayed in MATLAB.

In order to maximize the download speed for high data volume AMP files from the host PC that runs the monitor program to the DSP, a parallel interface connection has been implemented (see Figure 10 in Appendix 2). A high download speed is particularly important for smooth test administration using pre-processed AMP files, where the amount of data that needs to be downloaded can be on the order of 1 Mbyte or more. Before the parallel interface was available, data downloads at runtime from the host PC into the DSP memory were realized through a standard serial COM port connection. Because of the limited transfer rate of a serial connection (the maximum baud rate is 115200 bits per sec), only small data transfers were practical. The parallel connection interface allows a much higher data throughput. It connects the PC LPT1 port, operated in ECP mode to the HI32 host port on the DSP56301, which is operated in general-purpose I/O (GPIO) mode and serviced in the DSP code through polling. The parallel interface DSP code has been implemented as described in the MOTOROLA Application Note AN2085, "ECP Standard Parallel Interface for DSP56300 Devices", Rev.0, 11/2000.

User interface

In the streaming mode, the monitor is responsible for downloading the DSP data, including the stimulation pulse sequence and the DSP channel data tables. It also provides an interface for the administration of speech reception tests with consonants, sentences, and other materials.

Downloading patient specific information

A speech processor specification file, similar to the ones used to load real-time speech processors (see section "Setup and control of unilateral and bilateral speech processors in the monitor" in Appendix 2), is used to read patient specific data. Spec files for the real-time mode can also be used in the streaming mode, though the streaming mode will not use all the information in the spec file. The streaming mode requires:

- Implant type
- Channel information including electrode, threshold, MCL, and pulse width
- Rate of stimulation
- Synchronization mode for bilateral processors
- Channel order table

In addition to the channel data tables, patient thresholds, and MCL information, the streaming mode also loads a background stimulation data table. This is to simulate the constant stimulation that the subject receives in a real time processor even though (s)he is receiving no audio input. A dedicated background balance user interface controls the volume of the background pulses. Background volume for both left and right side is scaled according to the balance setting and the stimulus volume for the respective side (see the discussion of the streaming mode design for details). Once a test is started, the background pulses continue to be presented, even when loading a new AMP file, as long as the spec file is not changed.

The streaming mode can run in two ways – in the debug mode or using a streamplan file. In the debug mode, several AMP files to be downloaded into the DSP are added at the end of the speech processor spec file. When this is the case, the AMP files are loaded one at a time in sequence and the operator can repeat the currently loaded one as needed, or instruct the monitor to load and execute the next one. More frequently, the operator opens a "streamplan" text file, which has been prepared ahead of time. The streaming mode form displays the streamplan file and the patient specification file in separate memo pages as shown in Figure 4. The red highlighted line in the streaming plan page indicates the present condition that the monitor is loading or that is being tested. Each line in the streamplan file contains all the information needed to run one test. Each line includes:

- Patient identification
- Speech processor specification filename and path
- Directory where AMP files are stored
- Test information:
    - Noise condition: quiet, front, left, or right
    - Male or female talker
    - Type of test:
        - Medial consonant identification test: 16 English, 24 English, or 16 German
        - CUNY or HINT English sentences or OLSA German sentences [Oldenburger Satztest]

Once the streamplan file is open, the monitor will open the speech processor spec file and attempt to load the stimulation pulse sequence files. The monitor needs to verify that each AMP file loaded is compatible with the speech processor specifications. It verifies that both have the same specifications for:

- Synchronization settings for bilateral condition
- Number of channels
- Order of stimulation
- Rate of stimulation
- Implant interface

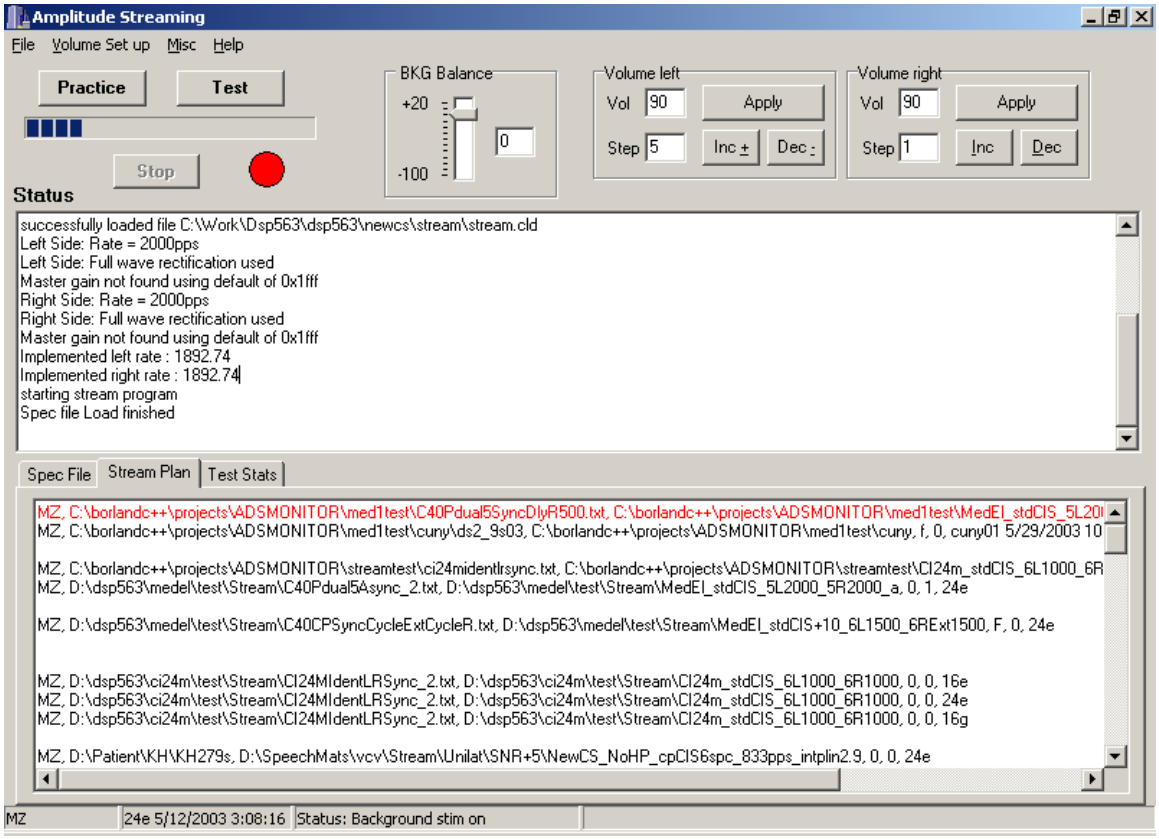


Figure 4: The streaming mode user interface: The line shown in red in the stream plan memo page identifies the currently loaded test

Consonant tests

When running a consonant test, the monitor will verify whether all the token AMP files fit into the DSP's memory at once. There are 3 exemplars for each consonant, so for a bilateral speech processor in a 24 consonant test, it will try to download 144 files into DSP memory. If all the tokens do not fit into memory, it will load one at a time during the patient testing procedure. If all tokens fit into memory, the monitor keeps a record of the memory address where each token starts, together with its length. It will need this information to instruct the DSP to properly play each token individually. For a bilateral processor, the monitor instructs the DSP to play either the same or two different stimulation pulse sequences on the two sides, depending on whether a single- or dual-sequence AMP file is encountered (see specification of AMP file format on p. 23).

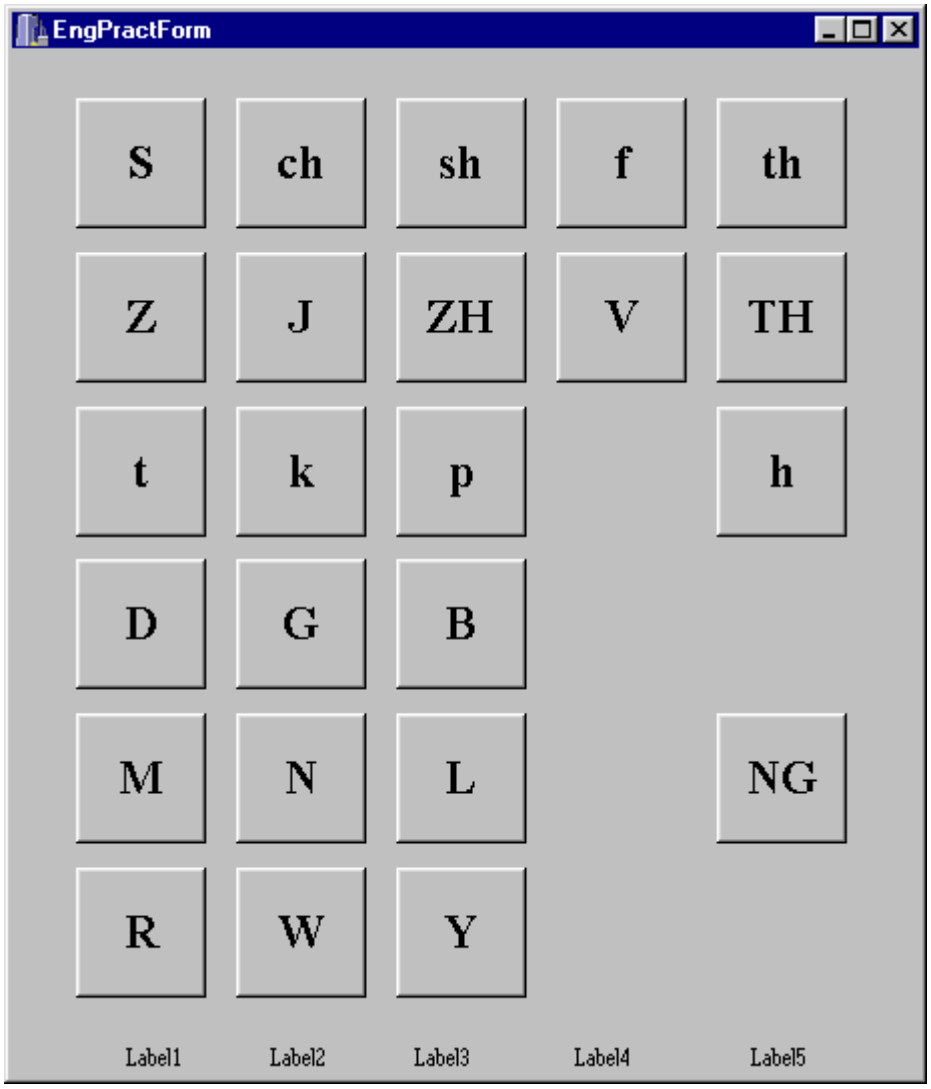


Figure 5: Subject practice display for a 24 consonant test

The subject can use the practice form to listen to consonants in order to familiarize her/himself with the task or to allow the volume to be set at a comfortable level. That form appears on a second monitor, leaving the control window available to the operator for changes in volume, if necessary. This is essentially the same form the subject will encounter during the test (see Figure 5), but here (s)he has command of what is played. The subject clicks on a button in the form and the consonant corresponding to that button is played. When the subject is finished practicing, the form is closed and the test can be started.

For the consonant test another form is opened, almost identical to the one used for practice. The monitor uses the same randomization files used for consonant tests with real-time processors, randomizing among tokens and exemplars with the tokens presented in sets to facilitate statistical analysis. The test is completed when each token has been played 5 times. The control window has a Test Stats memo page in which the operator can monitor the subject's performance. It displays the token presented, the subject's

answer, total number of tokens presented, and percent correct thus far. The monitor logs all the subject's answers into a result text file, which will later be used by other programs to run analysis on the data.

Once the test is completed, the monitor will write the test information and date into a text file named "streamdone", which maintains a complete history of all the streaming mode tests that were run. The current line in the streaming plan form is then erased and the monitor analyzes the next line, checking whether new information or new AMP files need to be loaded. The monitor is then ready to run a new test.

Sentence tests

The streamplan file defines the type of sentence test that will be run and the list number to use. Each list has 10-12 sentences (depending on the material used), which are loaded and played sequentially. Practice sentences are used for volume set-up. For this test the subject does not need a display. The operator scores the test by hand and has control of the progression of the test. The sentence AMP file downloaded is displayed in the status window. The test is finished when the monitor has loaded all the sentences in the list. It will write the test information and date into the streamdone text file, and proceed to the next line in the streamplan file.

Processing of digitized speech recordings in MATLAB

MATLAB program MakeAMP.m

Speech recordings in digital wave file format are processed in MATLAB using the script MakeAMP.m. This program converts monophonic or stereophonic wave file recordings into single- or dual-sequence amplitude sequence files (AMP files) for unilateral or bilateral presentation in the streaming mode. Digitized speech signals retrieved from a wave file are processed according to one of several different speech processing strategies. The resulting multi-channel output signals are sampled into a linear sequence of pulse amplitudes at the specified stimulation rate by sampling the parallel output channels in an interleaved fashion across channels in the specified stimulation order. MakeAMP scans basic stimulation parameters (such as unilateral or bilateral condition, number of channels, pulse rate, and stimulation order) from a speech processor spec file, like those used in the monitor program to set-up and run speech processors in real-time. The type of speech processing strategy and related parameters are read from the command line, if not provided as input arguments to MakeAMP. In the case of bilateral processing, independent sets of strategy parameters, including different speech processing strategies, are supported on the left and right side.

An important feature of MakeAMP is the ease with which it can be extended to integrate new processing strategies or implement variations of existing ones. Adding a new processing strategy merely requires adding a new strategy function call to MakeAMP. A strategy function is a MATLAB program that takes a signal vector and its sampling rate as input arguments and returns the channel output signals as output arguments. Strategy parameters are usually entered from the command line, if not provided as optional input arguments to the strategy function.

During processing, MakeAMP logs all parameters, such as signal processing strategy and strategy parameters, stimulation parameters scanned from a spec file, the spec file name, and input wave files processed to a log-file. For each AMP file being generated, the log-file also records signal duration in ms, maximum normalized channel output amplitude and rms-amplitude, number of pulses, and number of 24-bit DSP words of the resulting binary pulse amplitude sequence on each side.

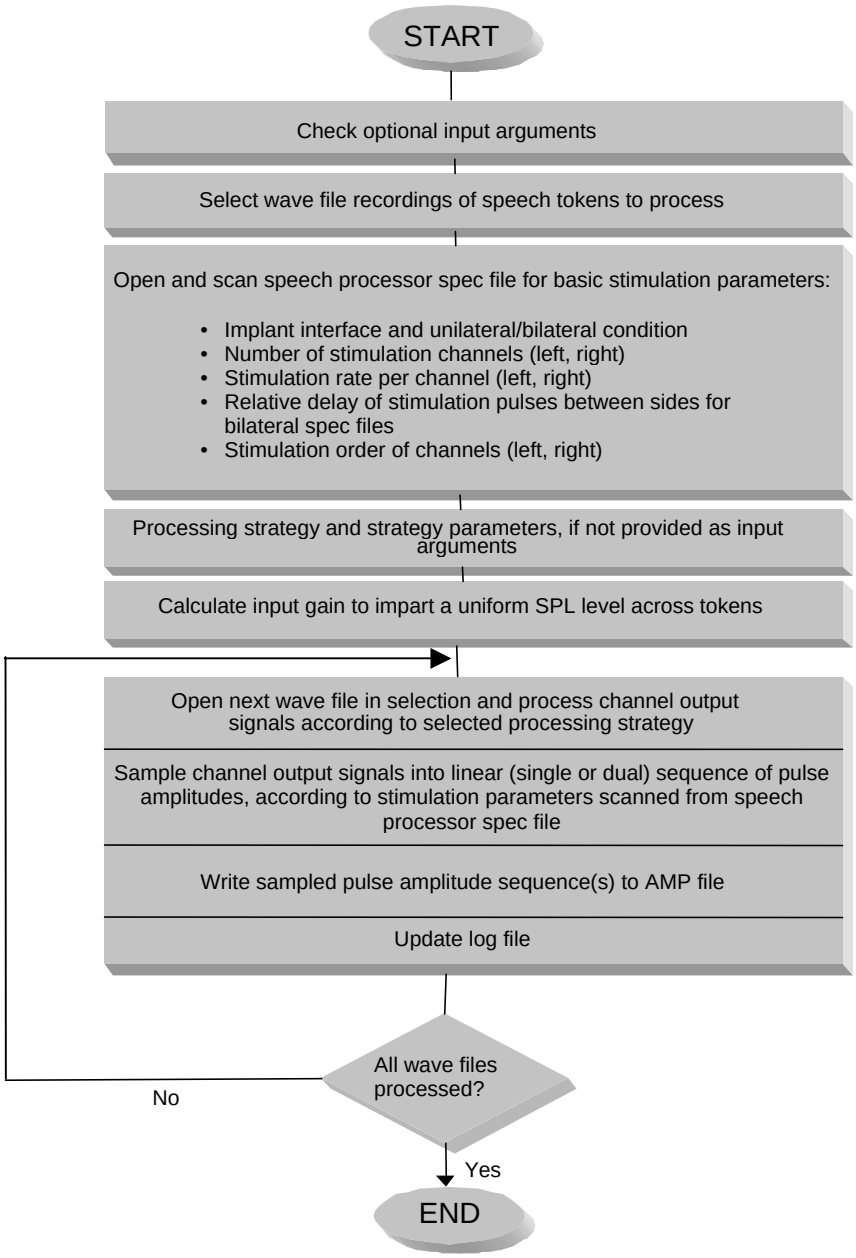


Figure 6: Flowchart of the processing steps in MakeAMP.m. A program listing is provided in Appendix 3.

For a unilateral condition (unilateral speech processor spec file), a single pulse amplitude sequence is generated from each wave file input and stored to an AMP file (if stereo information is included, either the left or right signal channel must be selected). Generally AMP files bear the same file name as the wave files from which they are derived, except that file extensions .WAV are replaced with .AMP.

For a bilateral condition (bilateral speech processor spec file), either single or dual pulse amplitude sequences can result, depending on wave file inputs and processing parameters. True stereo wave files with different signals on left and right channel (dichotic input conditions) always result in two linear amplitude sequences, the first one for the left implant and the second one for the right implant. For monophonic wave file signals or stereo wave files with identical signals on the left and right channel (diotic input conditions), the resulting two pulse amplitude sequences for the left and right implants are identical if the following conditions are satisfied:

1. Same processing strategy and identical strategy parameters on the left and right side
2. Same number of channels on the left and right side
3. Same stimulation order on the left and right side
4. Same pulse rate per channel on the left and right side
5. Synchronized stimulation pulses, with no relative offset between the two sides

Conditions 1 and 2 are sufficient for channel output signals to be identical on the left and right side and conditions 2-4 guarantee that the carrier pulses, which are modulated in amplitude with the channel output signals, occur simultaneously on the two sides for the whole stimulus duration. An essential requirement is also that pulse amplitudes as sampled in MakeAMP are normalized and that the linear scaling to the electrode specific THR and MCL levels occurs in the DSP.

When the pulse amplitude sequences for left and right implants are identical, MakeAMP stores only one sequence in the AMP file. This saves download time and memory space in the DSP. When executed bilaterally, the sequence of normalized pulse amplitudes is retrieved twice in the DSP, for dynamic range scaling on left and right side.

Table 1 summarizes the cases for unilateral and bilateral processing conditions and the resulting single-sequence or dual-sequence AMP files.

Implant interface	**Speech processor spec file**	**Wave file input**	**AMP file generated by MakeAMP.m**
Current Sources, Med-El, CI24M	Unilateral	Mono	Single-sequence
Current Sources, Med-El, CI24M	Unilateral	Stereo. Channel signal to be processed has to be specified.	Single-sequence
Med-El, CI24M	Bilateral	True stereo	Dual-sequence
Med-El, CI24M	Bilateral	Mono or stereo with identical signals on the left and right channels	Single-sequence, if strategy parameters and stimulation parameters (number of channels, rate, order, rdelay = 0 μs) on the left and right side are identical

Med-El, CI24M	Bilateral	Mono or stereo with identical signals on the left and right channels	Dual-sequence, if strategy parameters or stimulation parameters (number of channels, rate, order, rdelay ≠ 0 μs) on the left and right side are different

Table 1: AMP files resulting from different speech processor spec files and wave file inputs

Specification of AMP file format

AMP files, as generated in MakeAMP, are mixed ASCII text and binary files. They consist of a leading ASCII header section, followed by a single or dual binary amplitude sequence section.

The ASCII header section consists of a sequence of tags. Tags start with a '>' character, followed by a tag identifier. One or more lines containing the tag data for that tag follow a tag line. Vector data are specified one element per line. Data lines are delimited by the next tag line or by the end of the file. Comment lines can be inserted anywhere in the header by starting the line with an asterisk.

The ASCII header section terminates with the tag ">start bin", which designates the starting point of the binary amplitude sequence. The ">start bin" tag includes a CR (0x0D) and LF (0x0A) character, *i.e.* the binary amplitude sequence does not start earlier than two bytes after the tag string.

AMP files containing a single or dual pulse amplitude sequence have a different format, which is specified below.

Single-sequence AMP files

`>system`
(Implant interface string. Can be either "`new_currents`", "`medel`", or "`ci24m`")
`>length`
(Stimulus length in ms, rounded to an integer)
`>pulsecount`
(Number of pulses. Applies to both left and right sides in case of bilateral presentation)
`>implrate`
(Implemented stimulation rate in pulses per second, per channel, rounded to an integer. Applies to both sides in case of bilateral presentation)
`>order`
(Stimulation order of channels. Applies to both sides in case of bilateral presentation)
`>cyctim`
(Pulse onset times in μs for each pulse in stimulation cycle, relative to cycle start. Numbers are floats, rounded to 2 decimals. Applies to both sides in case of bilateral presentation)
`>start bin`
(Single pulse amplitude sequence in binary format. Binary sequence contains unsigned 12-bit amplitudes in packed, high bit first format, *i.e.* 2 amplitude values fill up 3 bytes. For an odd pulsecount, 12 zero-bits are added to make a 24-bit DSP DRAM word full)

Dual-sequence AMP files

>system
(Implant interface string. Can be either "medel" or "ci24m")
>length
(Stimulus length in ms, rounded to integer number)
>lpulsecount
(Number of pulses on left side)
>rpulsecount
(Number of pulses on right side)
>limplrate
(Implemented stimulation rate per channel on left side, rounded to integer number)
>rimplrate
(Implemented stimulation rate per channel on right side, rounded to integer number)
>lorder
(Stimulation order of channels on left side)
>rorder
(Stimulation order of channels on right side)
>lcyctim
(Pulse onset times in μs for each pulse in stimulation cycle on left side, relative to cycle start. Numbers are floats, rounded to 2 decimals)
>rcyctim
(Pulse onset times in μs for each pulse in stimulation cycle on right side, relative to cycle start. Numbers are floats, rounded to 2 decimals)
>rdelay
(Delay of stimulation cycle start on right side in μs, relative to cycle start on left side. Delay is positive (right delayed versus left) or negative (left delayed versus right) float, rounded to 2 decimals)
>start bin
[Binary pulse amplitude sequence for left side. Byte sequence is zero-filled to make last 24-bit DSP DRAM word full]
[Binary pulse amplitude sequence for right side. Byte sequence is zero-filled to make last 24-bit DSP DRAM word full]

Note: Due to implant pulse timing constraints, the pulse rate per channel that can actually be implemented in a given implant system might differ slightly from the rate specified in the speech processor spec file that is scanned in MakeAMP. Thus, the implant interface system and the implemented (as opposed to the specified) pulse rate are inserted into the AMP file header by MakeAMP. When the monitor program downloads an AMP file, it accepts only files in which the implemented rate, the number of channels, and the stimulation order match the rate, number of channels, and stimulation order set up by the monitor when starting the background stimulation for embedding stimuli whose pulse amplitudes are retrieved from downloaded AMP files. This crosscheck in the monitor program is an important safety feature. It ensures that an AMP file can never be streamed out to the subject implant(s) at an incompatible rate or stimulation order, or on a wrong number of electrodes (this would result in a potentially hazardous distortion of the stimulus).

Example of a single-sequence AMP file and its log-file

The following example shows the ASCII header of a single-sequence AMP file for the medial consonant utterance 'ASA' in quiet (file M0510.amp), as generated by MakeAMP from a monophonic wave file input from file M0510.wav. The input spec file is for a bilateral 8-channel Med-El processor. Strategy and stimulation parameters are identical on the left and right sides, thus the same amplitude sequence results on both sides (case 4 in Table 1). Figure 3 shows oscilloscope traces of the pulse train envelope signals on channels 1 and 8 of AMP file M0510.amp, as streamed to a Med-El C40+ implant detector.

```
* Amplitude sequence file for streaming mode on ADS56301 Lab System.
* File contains unilateral amplitude sequence and was generated
* by MakeAMP.m version 2.4.
* Input wave file name:
*   E:\AudioData\vcv\hrtf\Mixed_ramp_+5dB\M0510.wav
* Input wave file mode:
*   Stereo with identical signals on left and right channel -> Mono
* Signal processing strategy:
*   stdCIS
* ADSMonitor speech processor spec file scanned for processor parameters:
*   C:\rtisys\SpecFiles\DS2\DS2_1
* Spec file parameters:
*   Number of channels: 8 left
*                       8 right
*   Stimulation rate per channel: 1515 pps left
*                                 1515 pps right
*   Stimulation order:  1  5  2  6  3  7  4  8 left
*                       1  5  2  6  3  7  4  8 right
*   Synchronization mode: sync, no delay between sides
* Spec file results in identical nchn, order, implRate and cyctim
* (with rdelay = 0) on both left and right side. Thus, this AMP file,
* generated from a wave file in mono input mode, is unilateral.
*
>system
MedEl
* Stimulus duration in msecs
>length
863
* Number of deterministic pulse amplitudes in stimulus
>pulsecount
10464
* Implemented rate in pps per channel
>implrate
1515
>order
1
5
2
6
3
7
4
8
* Onset times of pulses within stimulation cycle in us, relative to cycle start
>cyctim
0.00
81.67
163.33
245.00
326.67
410.00
493.33
576.67
* Pulse amplitude sequence in binary format
>start bin
```

The contents of the log-file pertaining to the AMP file header shown above (from file M0510.amp) are shown below (only the line logged for processing of AMP file M0510.amp is shown at the end):

```
* Log file of amplitude sequence files generated for streaming mode
* on ADS56301 Lab System.
* Files were generated by MakeAMP.m version 2.4.
*
* Input directory for wave files:
*   E:\AudioData\vcv\hrtf\Mixed_ramp_+5dB\
*
* Output directory for amp files:
*   E:\AudioData\vcv\hrtf\Mixed_ramp_+5dB\MedEl_stdCIS_8L1515_8R1515_s\
*
* ADSMonitor speech processor spec file scanned for processor parameters:
*   C:\rtisys\SpecFiles\DS2\DS2_1
*
* Implant interface specification in spec file:
*   MedEl
*
* Spec file parameters:
*   Number of channels: 8 left
*                       8 right
*   Stimulation rate per channel: 1515 pps left
*                                 1515 pps right
*   Stimulation order:  1  5  2  6  3  7  4  8 left
*                       1  5  2  6  3  7  4  8 right
*   Synchronization mode: sync, no delay between sides
*
* Signal processing strategy:
*   stdCIS left and right side
*
* Strategy parameters for left and right side:
*   inputGain    : 1.35869
*   LeqRef       : -9
*   nchn         : 8
*   eqcutoff     : 1200
*   eqord        : 1
*   fl           : 350
*   fu           : 5500
*   bpord        : 6
*   gainBPout    : 1
*   lpcutoff     : 200
*   lpord        : 4
*   rect         : 'full'
*   gainMAPin    : 4
*   pwr          : -0.0001
*   thr          : 0
*   mcl          : 1
*   Nmaplaw      : 1024
*   intp         : 'on'

Wave file -> Amp file (duration, max chnl output amp/rms, no. of pulses, no. of DSP words)
-------------------------------------------------------------------------------------------
...
> Stereo wave file M0510.wav has identical signals on left and right channel and will be processed in
mono mode.
M0510.wav -> M0510.amp (863 ms, 0.924/0.557, 10464 = 0x28E0 pulses, 5232 = 0x1470 DSP words)
...
```

Streaming of AMP files in the monitor program with a speech processor spec file

Table 2 shows possible combinations of speech processor spec files and AMP files. Generally, a given AMP file can be used on all implant interfaces where number of channels (unilateral or bilateral), implemented pulse rate, stimulation order, and synchronization settings (for bilateral interfaces only) match between the AMP file and the speech processor spec file through which background stimulation for embedding the stimulus pulse sequence was set up in the monitor program.

Implant interface	Speech processor spec file	AMP file	Safety check in monitor program
Current Sources, Med-El, CI24M	Unilateral	Single-sequence	Execute unilateral stimulus from pulse amplitude sequence, if number of channels, implemented rate, and stimulation order match
Med-El, CI24M	Unilateral	Dual-sequence	Execute unilateral stimulus from pulse amplitude sequence on side specified in unilateral spec file, if number of channels, implemented rate, and stimulation order match
Med-El, CI24M	Bilateral	Single-sequence	Execute bilateral stimulus by playing same pulse amplitude sequence on both left and right side, if number of channels, implemented rate, and stimulation order match and if pulses occur simultaneously on left and right side (i.e. same pulse onset times on both sides and no relative delay).
Med-El, CI24M	Bilateral	Dual-sequence	Execute bilateral stimulus from pulse amplitude sequences if number of channels, implemented rate, relative delay (i.e. pulse onset times), and stimulation order match on left and right side

Table 2: Possible combinations of speech processor spec files and AMP files in the streaming mode

Signal processing strategy implementations in MATLAB

A number of signal processing strategies, including a standard CIS strategy, have been implemented in MATLAB. Table 3 shows the currently available strategy functions and their basic design features. All of these strategy functions can be called from within MakeAMP.

Strategy function name	Properties
stdCIS	Standard CIS strategy, consisting of the following processing blocks: Pre-emphasis high-pass filter Bank of Butterworth band-pass filters Full- or half-wave rectifier and low-pass filter as envelope extractor Instantaneous power maplaw compression

stdihcCIS	CIS strategy, where envelope extraction stage is replaced by processing stage that implements the inner hair cell (IHC) model (Meddis, 1986; Meddis, 1988): Pre-emphasis high-pass filter Bank of Butterworth band-pass filters Meddis IHC model stage as envelope extractor Instantaneous power maplaw compression (typically reduced compression)
stdihclpCIS	CIS strategy, where envelope extraction stage is replaced by the Meddis IHC model and a subsequent low-pass filter stage: Pre-emphasis high-pass filter Bank of Butterworth band-pass filters Meddis IHC model stage, followed by low-pass filter, as envelope extractor Instantaneous power maplaw compression (typically reduced compression)
stdenvihcCIS	CIS strategy, where envelope extraction stage is composed of rectifier and low-pass filter, followed by Meddis IHC model: Pre-emphasis high-pass filter Bank of Butterworth band-pass filters Full- or half-wave rectifier, low-pass filter and Meddis IHC model stage as envelope extractor Instantaneous power maplaw compression (typically no or reduced compression)
stdhpCIS	CIS strategy, where envelope extraction stage is composed of a rectifier and low-pass filter, followed by a high-pass stage designed to enhance signal onsets and simulate auditory nerve-like adaptation properties: Pre-emphasis high-pass filter Bank of Butterworth band-pass filters Full- or half-wave rectifier, low-pass filter, and high-pass adaptation stage Instantaneous power maplaw compression
cpCIS	CIS strategy, where Butterworth band-pass filter bank is replaced by a bank of nonlinear DRNL filters (Meddis, O'Mard, Lopez-Poveda, 2001; Lopez-Poveda, Meddis, 2001): Pre-emphasis high-pass filter Bank of nonlinear DRNL band-pass filters Full- or half-wave rectifier and low-pass filter as envelope extractor Instantaneous power maplaw compression (typically reduced compression)
cpihcCIS	CIS strategy with bank of nonlinear DRNL filters and Meddis IHC model as envelope extraction stage: Pre-emphasis high-pass filter Bank of nonlinear DRNL band-pass filters Meddis IHC model stage as envelope extractor Instantaneous power maplaw compression (typically reduced compression)
cpihclpCIS	CIS strategy with bank of nonlinear DRNL filters and Meddis IHC model/low-pass filter as envelope extraction stage: Pre-emphasis high-pass filter Bank of nonlinear DRNL band-pass filters Meddis IHC model stage, followed by low-pass filter, as envelope extractor Instantaneous power maplaw compression (typically reduced compression)

cpenvihcCIS	CIS strategy with bank of nonlinear DRNL filters and envelope extraction stage, composed of rectifier and low-pass filter, followed by Meddis IHC model: Pre-emphasis high-pass filter Bank of nonlinear DRNL band-pass filters Full- or half-wave rectifier, low-pass filter and Meddis IHC model stage as envelope extractor Instantaneous power maplaw compression (typically linear, applying only output gain)

Table 3: Strategy functions implemented in MATLAB

All strategy implementations in Table 3 can be called as functions in MakeAMP. A signal vector, sampling rate, and strategy parameters are provided as input arguments and a matrix of channel output signals is returned as the output argument.

Strategy functions can also be run as standalone programs, by simply executing the function directly within the MATLAB command environment. In this case, an input wave file and strategy parameters are requested interactively and all signals from the input signal through the various processing stages to the final channel output signals are plotted in a sequence of graphs. Also, signal rms-amplitudes and [min, max] amplitude ranges are displayed in the MATLAB command window throughout the processing stages. As an additional tool for signal analysis, all plot windows have the capability of zooming into signal segments and plotting and analyzing signal spectrograms.

Plot windows generated by strategy functions executed in standalone mode also provide buttons for displaying spectrograms and for playing signals at the output of all processing stages as sound. Another feature of the standalone execution mode of strategy functions is the ability to play signals. In the case of channel output envelope signals, a paradigm of acoustically simulating speech reception with cochlear implants is applied. Bands of noise, amplitude-modulated by the respective channel envelope signals, are added together and played as sound. Noise bands are derived by filtering white noise, through the same band-pass filter that is used in the signal processing strategy, to derive the channel output envelope signal of that particular channel.

The following figures show the plots generated in stdCIS when run in the standalone mode for the medial consonant input 'ASA' from wave file M0510.wav. Signal rms-amplitudes and [min, max] amplitude ranges as displayed in the MATLAB command window throughout all processing stages are also shown. Processing parameters are 8 channels, with a 1^{st}-order high-pass at cutoff frequency of 1200 Hz, 8 equally-spaced band-pass filters of 6^{th} order on a logarithmic scale from 350 to 5500 Hz, full-wave rectification, a 4^{th}-order low-pass filter at cutoff frequency 200 Hz for envelope extraction, and a maplaw compression power of –0.0001. Note the match between the compressed channel output signals for channels 1 and 8 in Figure 9 (black) and the stimulus pulse train envelopes on channels 1 and 8 in Figure 3. The AMP file used to generate the stimulation waveforms in Figure 3 has been derived from the same input wave file M0510.wav, using the strategy and parameters indicated above.

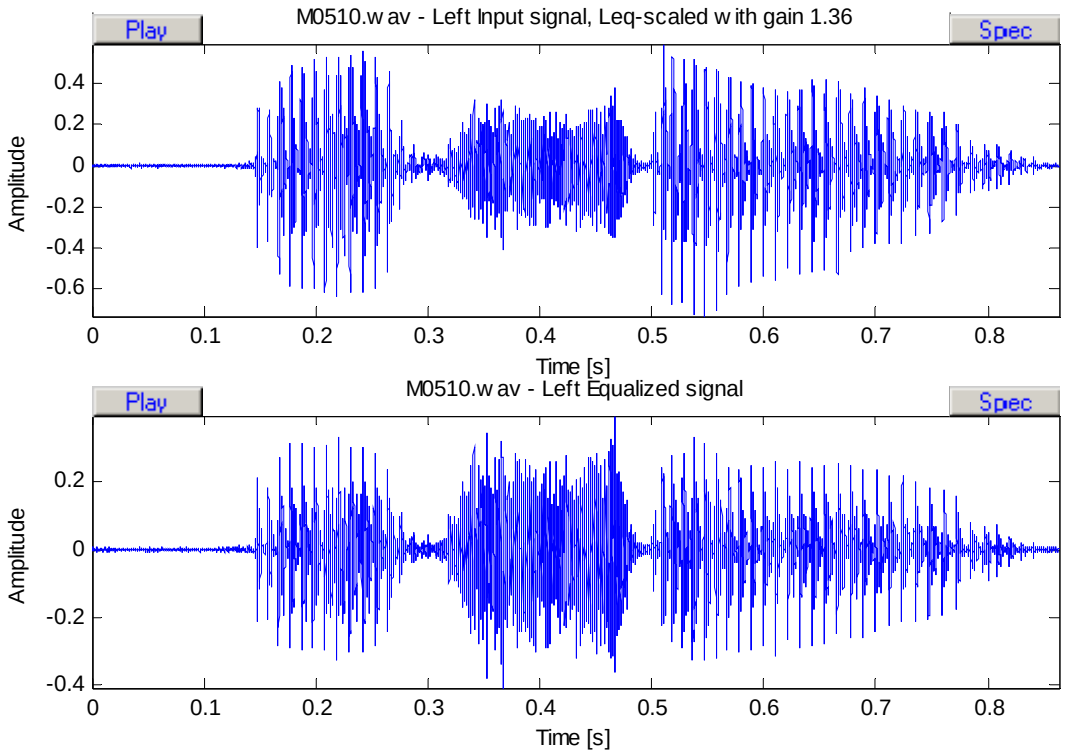


Figure 7: Plot of input signal 'ASA' from file M0510.wav (top) and corresponding output signal of pre-emphasis high-pass filter (bottom)

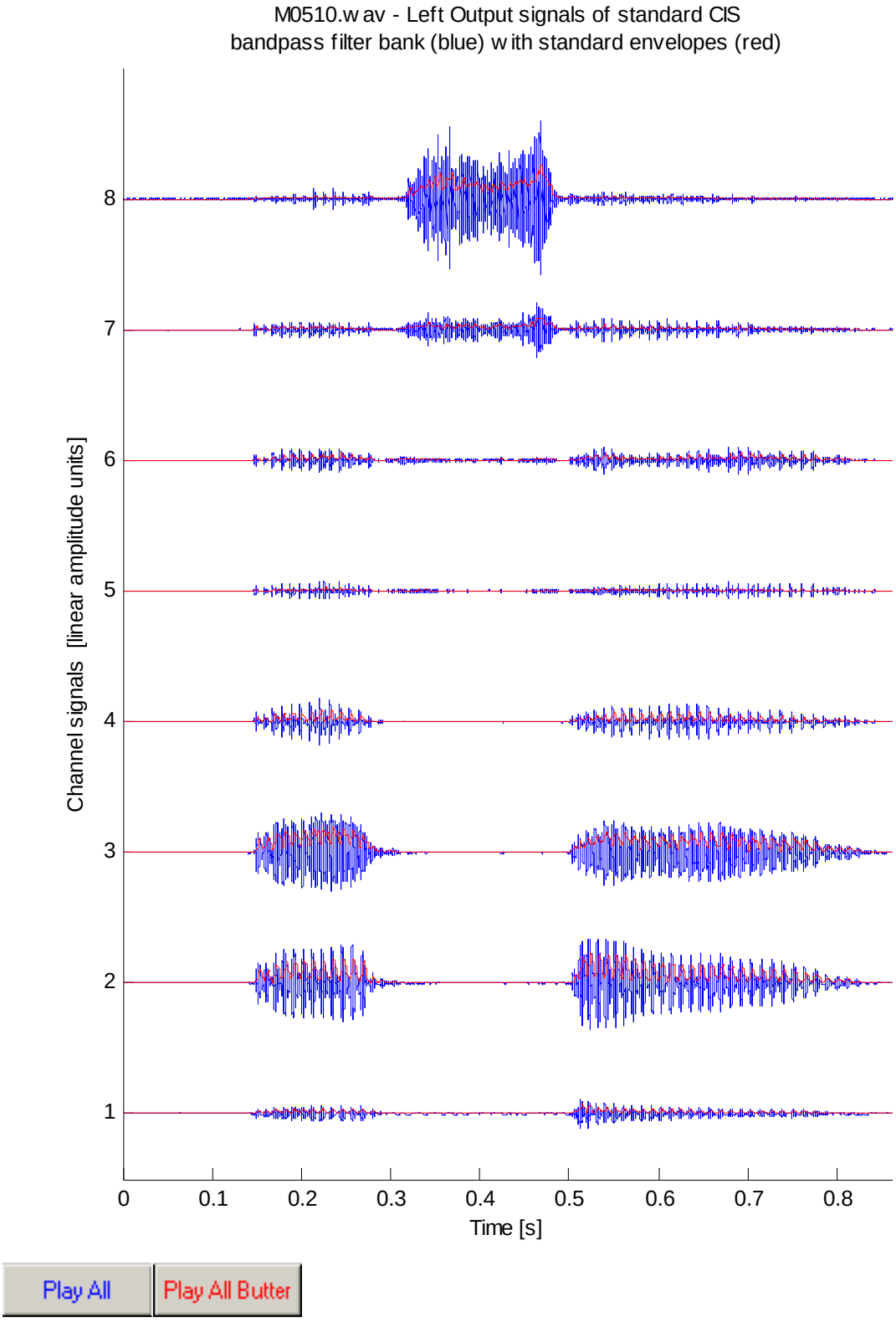


Figure 8: Bandpass filter output signals (bipolar, blue) and corresponding envelopes (unipolar, red) of channels 1 through 8 for medial consonant 'ASA'

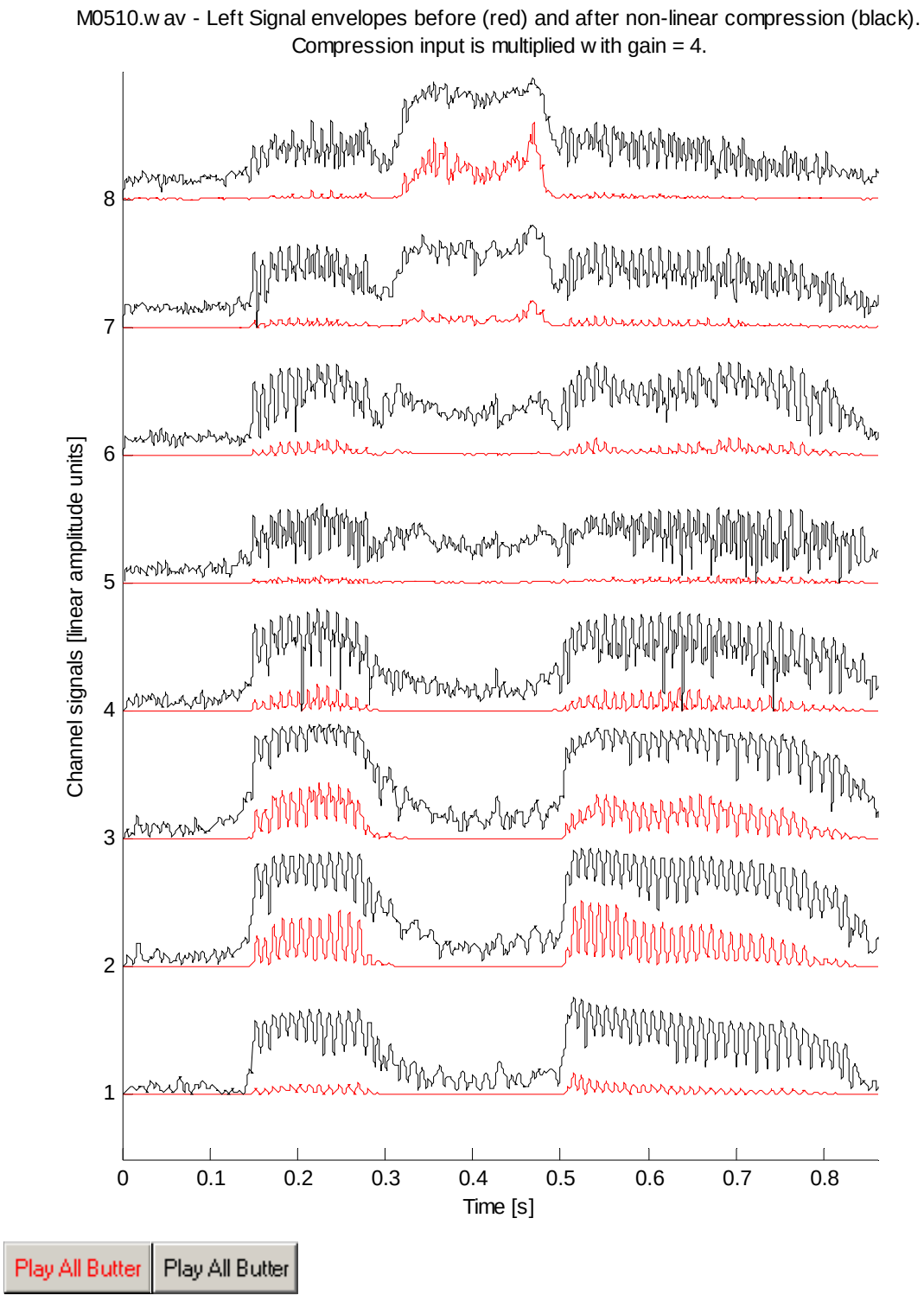


Figure 9: Envelope extraction output signals (lower, red) and compressed envelopes (upper, black) of channels 1 through 8 for medial consonant 'ASA'. See Figure 3 for the oscilloscope traces of channels 1 and 8 as captured during streaming of the AMP file that was derived from the same consonant in MATLAB.

The following shows the MATLAB command window output, as generated during the standalone execution of strategy function stdCIS with input wave file M0510.wav and parameters taken from strategy parameter structure stratpar{1}. The output shows signal rms-amplitudes and [min, max] amplitude ranges through all processing stages of the input signal.

```
>> stratpar{1}

ans =

    inputGain: 1.3587
       LeqRef: -9
         nchn: 8
     eqcutoff: 1200
        eqord: 1
           fl: 350
           fu: 5500
        bpord: 6
    gainBPout: 1
     lpcutoff: 200
        lpord: 4
         rect: 'full'
    gainMAPin: 4
          pwr: -1.0000e-004
          thr: 0
          mcl: 1
      Nmaplaw: 1024
         intp: 'on'

>> stdcis(stratpar{1})

Processing input wave file 'M0510.wav' at fs = 44.1 kHz according to standard CIS strategy...
Stereo input signal specified. Selected left channel for processing.
DC compensating and Leq-scaling input signal with gain 1.36.
Equalizing signal with Butterworth highpass filter of order 1 and cutoff frequency 1200 Hz.
Analyzing signal with bank of 8 Butterworth bandpass filters of order 6 and range 350-5500 Hz.
Channel 1: Butterworth bandpass filter at [350, 494] Hz
     Filter input signal range:  [  -0.4089548,   0.3972495 ], rms: 0.0745608
     Filter output signal range: [  -0.0652551,   0.0616001 ], rms: 0.0094280

Channel 2: Butterworth bandpass filter at [494, 697] Hz
     Filter input signal range:  [  -0.4089548,   0.3972495 ], rms: 0.0745608
     Filter output signal range: [  -0.1958718,   0.1900900 ], rms: 0.0411206

Channel 3: Butterworth bandpass filter at [697, 983] Hz
     Filter input signal range:  [  -0.4089548,   0.3972495 ], rms: 0.0745608
     Filter output signal range: [  -0.1621509,   0.1670289 ], rms: 0.0417575

Channel 4: Butterworth bandpass filter at [983, 1387] Hz
     Filter input signal range:  [  -0.4089548,   0.3972495 ], rms: 0.0745608
     Filter output signal range: [  -0.0963249,   0.1000811 ], rms: 0.0139269

Channel 5: Butterworth bandpass filter at [1387, 1958] Hz
     Filter input signal range:  [  -0.4089548,   0.3972495 ], rms: 0.0745608
     Filter output signal range: [  -0.0346579,   0.0400986 ], rms: 0.0053111

Channel 6: Butterworth bandpass filter at [1958, 2762] Hz
     Filter input signal range:  [  -0.4089548,   0.3972495 ], rms: 0.0745608
     Filter output signal range: [  -0.0595480,   0.0589883 ], rms: 0.0100559

Channel 7: Butterworth bandpass filter at [2762, 3898] Hz
     Filter input signal range:  [  -0.4089548,   0.3972495 ], rms: 0.0745608
     Filter output signal range: [  -0.1158118,   0.1162614 ], rms: 0.0118430

Channel 8: Butterworth bandpass filter at [3898, 5500] Hz
     Filter input signal range:  [  -0.4089548,   0.3972495 ], rms: 0.0745608
     Filter output signal range: [  -0.3173908,   0.3308889 ], rms: 0.0347280

Extracting signal envelopes with full-wave rectifier and
 Butterworth lowpass filter of order 4 and cutoff frequency 200 Hz.
 Channel output signal ranges after envelope extraction:
   Envelope signal range of channel 1:                        [  0.0000000, 0.0416566 ], rms: 0.0084312
   Envelope signal range of channel 2:                        [  0.0000000, 0.1267423 ], rms: 0.0369546
   Envelope signal range of channel 3:                        [  0.0000000, 0.1060866 ], rms: 0.0372608
   Envelope signal range of channel 4:                        [  0.0000000, 0.0526441 ], rms: 0.0119623
   Envelope signal range of channel 5:                        [  0.0000000, 0.0157263 ], rms: 0.0040980
   Envelope signal range of channel 6:                        [  0.0000000, 0.0324987 ], rms: 0.0087562
   Envelope signal range of channel 7:                        [  0.0000000, 0.0513290 ], rms: 0.0099987
   Envelope signal range of channel 8:                        [  0.0000000, 0.1469376 ], rms: 0.0290124
```

```
Applying gain = 4 to envelope signals before going into maplaw compression stage.
Compressing signal envelopes with power-maplaw p = -0.0001, N = 1024 and linear interpolation.
 Channel output signal ranges after maplaw compression:
   Maplaw output signal range of channel 1:                    [  0.0000000, 0.7423775 ], rms: 0.3781120
   Maplaw output signal range of channel 2:                    [  0.0000000, 0.9023051 ], rms: 0.5331152
   Maplaw output signal range of channel 3:                    [  0.0000000, 0.8767004 ], rms: 0.5572129
   Maplaw output signal range of channel 4:                    [  0.0000000, 0.7759691 ], rms: 0.4084730
   Maplaw output signal range of channel 5:                    [  0.0000000, 0.6032336 ], rms: 0.3427093
   Maplaw output signal range of channel 6:                    [  0.0000000, 0.7068038 ], rms: 0.4163818
   Maplaw output signal range of channel 7:                    [  0.0000000, 0.7723370 ], rms: 0.4256139
   Maplaw output signal range of channel 8:                    [  0.0000000, 0.9235915 ], rms: 0.4563326
Processing finished.
```

III. References

MOTOROLA, DSP 56300 Family Manual, DSP56300FM/AD, Rev. 3.0, 11/2000

Micea MV, Chiciudean D, Muntean L: ECP Standard Parallel Interface for DSP56300 Devices. MOTOROLA Application Note AN2085/D, Rev.0, 11/2000

Meddis R: Simulation of mechanical to neural transduction in the auditory receptor. *J. Acoust. Soc. Am.*, 79:702-11, 1986

Meddis R: Simulation of auditory-neural transduction: Further studies. *J. Acoust. Soc. Am.* 83:1056-63, 1988

Meddis R, O'Mard LP, Lopez-Poveda EA: A computational algorithm for computing nonlinear auditory frequency selectivity. *J. Acoust. Soc. Am.* 109:2852-61, 2001

Lopez-Poveda EA, Meddis R: A human nonlinear cochlear filter bank. *J. Acoust. Soc. Am.* 110:3107-18, 2001

IV. Plans for the next quarter

Among the activities planned for the next quarter are:

- Full implementation of the Advanced Bionics Corp. Bilateral CII Bionic Ear research interface for conducting bilateral speech reception and psychophysical studies with CII implant subjects. The research interface will provide access to the full capabilities of the CII implant system for parallel simultaneous pulsatile stimulation at high rates.

- Enhancement of the parallel multi-channel stimulation capabilities of the current source interface system. Development of an intermediate hardware interface between the ADS56301 signal processor and the current source interface to allow for parallel stimulation on up to 24 electrodes at high rates.

- Implementation of subject training procedures within the psychophysical task monitor programs.

- Continuing studies with local subjects ME16 and ME22, implanted bilaterally with Med-El Tempo+ devices.

- Three weeks of studies with return subject ME-15, July 14 – August 1.

- A return visit by consultant Enrique Lopez-Poveda from Salamanca, Spain, July 17-30, to coincide with the visit of subject ME-15.

- A visit by Craig Buchman, MD, Associate Professor, Chief, Division of Otology/Neurotology and Skull Base Surgery, UNC-CH, Chapel Hill, NC, August 1.

- Two weeks of studies with return subject ME-3, August 4-15.

- Attendance by Blake Wilson, Dewey Lawson, Lianne Cartee, Reinhold Schatzer, Xiaoan Sun and Robert Wolford at the 2003 Conference on Implantable Auditory Prostheses (CIAP 2003), Asilomar CA, August 17-22.

- A presentation by Blake Wilson on “Comparisons among new approaches for representing speech information with cochlear implants” at CIAP 2003.

- A poster presentation by Reinhold Schatzer on “A Novel CI speech processing structure for closer mimicking of normal auditory functions” at CIAP 2003.

- Two weeks of studies with return subject ME-10, August 25-September 5.

- Initial surgeries for at least two of the Nucleus Contour Electrode percutaneous study patients at Duke University Medical Center, Durham, NC.

- Two weeks of studies with return subject ME-19, September 22- October 3.

V. Acknowledgments

We thank volunteer research subjects ME-12, ME-14, ME-16, ME-18, and ME-22, who participated in studies conducted during this quarter.

Appendix 1: Summary of reporting activity for this quarter

Publications

Wilson BS, Lawson DT, Müller JM, Tyler RS, Kiefer J: Cochlear implants: Some likely next steps. *Annu Rev Biomed Eng* 5: 207-249, 2003.

Loeb GE, Wilson BS: Cochlear prosthesis. In Adelman G, Smith BH (eds.), *Encyclopedia of Neuroscience*, 3rd edition. Amsterdam: Elsevier.

Invited Presentations

Wilson BS, Brill SM, Cartee LA, Cox JH, Lawson DT, Schatzer R, Sun X, Wolford RD: Recent progress and likely next steps in the development of cochlear implants. *VII International Conference on Cochlear Implants and Related Audiological Sciences*, Warsaw – Kajetany, Poland, May 22-25, 2003.

Wilson BS, Lawson DT, Brill SM, Wolford RD, Schatzer R, Kiefer J, Pfennigdorff T, Pok M, Tillein J, Gstöttner W, Baumgartner W-D, Gilmer C, Pillsbury HC: Results from speech reception studies. Satellite Symposium on "Partial deafness cochlear implantation," *VII International Conference on Cochlear Implants and Related Audiological Sciences*, Warsaw – Kajetany, Poland, May 22-25, 2003.

Wilson BS, Brill SM, Cartee LA, Cox JH, Lawson DT, Schatzer R, Sun X, Wolford RD: Recent and future cochlear implant stimulation strategies. Conference celebrating *25 Years of Cochlear Implants in Vienna*, Vienna, Austria, June 19, 2003.

Chaired Sessions

Hochmair E, Wilson B, Lenhardt M, Czyzewski A (Co-Chairs): Session on "Cochlear and Brain Stem Implants and Related Audiological Problems. *VII International Conference on Cochlear Implants and Related Audiological Sciences*, Warsaw – Kajetany, Poland, May 22-25, 2003.

Wilson B, Skarzynski H (Co-Chairs): Satellite Symposium on "Partial deafness cochlear implantation." *VII International Conference on Cochlear Implants and Related Audiological Sciences*, Warsaw – Kajetany, Poland, May 22-25, 2003.

Appendix 2: Bilateral interfaces and control programs

In this appendix we provide a description of the hardware and software tools that have been developed and implemented over the course of the last several quarters to extend the functionality and optimize the performance of our speech laboratory system in terms of bilateral stimulation capabilities. These new tools significantly extend our ability to perform a variety of speech reception and psychophysical test measures our bilateral implant interfaces.

Design Objectives

1. Full control of bilateral Med-El C40/C40+, Nucleus CI22 and CI24M implants, by means of:

    - Bilateral, real-time CIS speech processor. Features include:

        - Ability to exactly synchronize pulsatile stimulation on both sides, either on a pulse-to-pulse basis, or synchronization of stimulation cycles over all channels on one side.

        - Specifiable relative offset of bilateral stimulation pulses with synchronized processors in multiples of the smallest time step supported by the implant.

        - Support for mono- and stereophonic audio input with bilateral stimulation.

        - Flexible way of controlling and modifying processor parameters, such as number of channels, filter characteristics, maplaw compression, rate, and subject specific threshold and most comfortable loudness levels, within and across channels.

    - Bilateral pulse train generation for psychophysical measurements. Features include:

        - Control of synchronization and relative delay of bilateral pulse bursts in multiples of the smallest time step supported by the implant, for both constant amplitude and sinusoidally amplitude modulated (SAM) pulse bursts.

        - Control of amplitude differences in bilateral pulse trains in multiples of the smallest amplitude step size supported by the implant.

        - User interface for generation of single pulse trains for basic psychophysical measurements.

        - User interface for administration of interaural timing difference (ITD) and interaural amplitude difference (IAD) tests. Support for interaural phase difference tests for bilateral SAM pulse trains.

Organization of the laboratory

Figure 10 shows a block diagram of the speech lab instrumentation setup. The heart of the laboratory system is a 100-MHz Motorola DSP56301 development board (ADS56301), which recently replaced a less powerful 66-MHz ADS56301 platform. Interface systems for the following implants are available and can be connected to the DSP board via a generic parallel bus interface:

- Bilateral interface for Nucleus CI22 implants
- Bilateral interface for Nucleus CI24M implants
- Bilateral interface for Med-El C40/C40+ implants
- Laboratory current source system for direct stimulation of implanted electrodes in subjects with percutaneous connectors

Recently, we have obtained the latest version of the Bilateral CII Bionic Ear research interface from Advanced Bionics Corp. Full implementation of that system will provide us with the capability to interface with all currently available clinical cochlear implant systems.

The ADS56301 is connected via 3 interfaces to a host PC running a monitor program, which downloads program codes and stimulation data/parameters and controls program execution in the Digital Signal Processor (DSP). The monitor program runs in a 32-bit Windows environment and implements a graphical interface for a variety of tasks and procedures under the test administrator's control, which will be described later. The OnCE port interface allows the monitor program to initialize the DSP with programs and data in its reset state, *i.e.* when no DSP program is running, and to start and stop DSP program execution. The RS-232 connection to the controlling PC's COM port is used as a byte-oriented two-way communication path with a software handshake between DSP and monitor at runtime, e.g. to dynamically modify the stimulation parameters such as volume setting in running speech processors. Finally, the parallel connection to the host PC's LPT port is a dedicated high-throughput data download channel from PC to DSP, that is used in the streaming mode at runtime to quickly download large pulse data sequences.

Currently, the DSP supports up to 3 different stimulation modes in conjunction with the various implant interfaces. These program modes are real-time speech processor emulators, psychophysics, and streaming mode. In the speech processor emulation mode, the DSP executes a CIS algorithm, processing the audio input signal to the ADC in either monophonic (for unilateral and bilateral speech processors) or stereophonic (bilateral speech processors only) input mode. As shown in Figure 10, audio input signals are fed to the DSP board (lab processor) through a serial connection from an off-board ADC subsystem. This subsystem is based on a commercial Burr-Brown DEM-PCM1760 evaluation fixture, containing an enhanced noise-shaping 20-bit stereo analog-to-digital converter. The ADC samples the analog input at a rate of 48 kHz per side. A wide variety of stereo and monophonic signal sources can be routed to the DSP analog inputs through a 14-channel audio mixing panel. These include standalone sources (operator headset microphone, room microphone, two audio cassette tape decks) as well as sources under the control of a second PC (video laser-disk player, audio CD-player, and embedded sound card). The second PC controls presentation of recorded or synthesized stimuli (*e.g.* speech tokens, noise) from these sources as well as video presentation of images from the video-laser disk for sound-plus-vision speech testing.

For psychophysical experiments, the DSP generates pulsatile stimulation patterns set up by the monitor program in DSP memory under operator control. In the streaming mode, the monitor downloads off-line generated stimulation sequences to DSP memory, for subsequent delivery by the DSP. Stimulation sequences typically are derived from speech test materials in digital format, such as a set of medial consonant or sentence audio files.

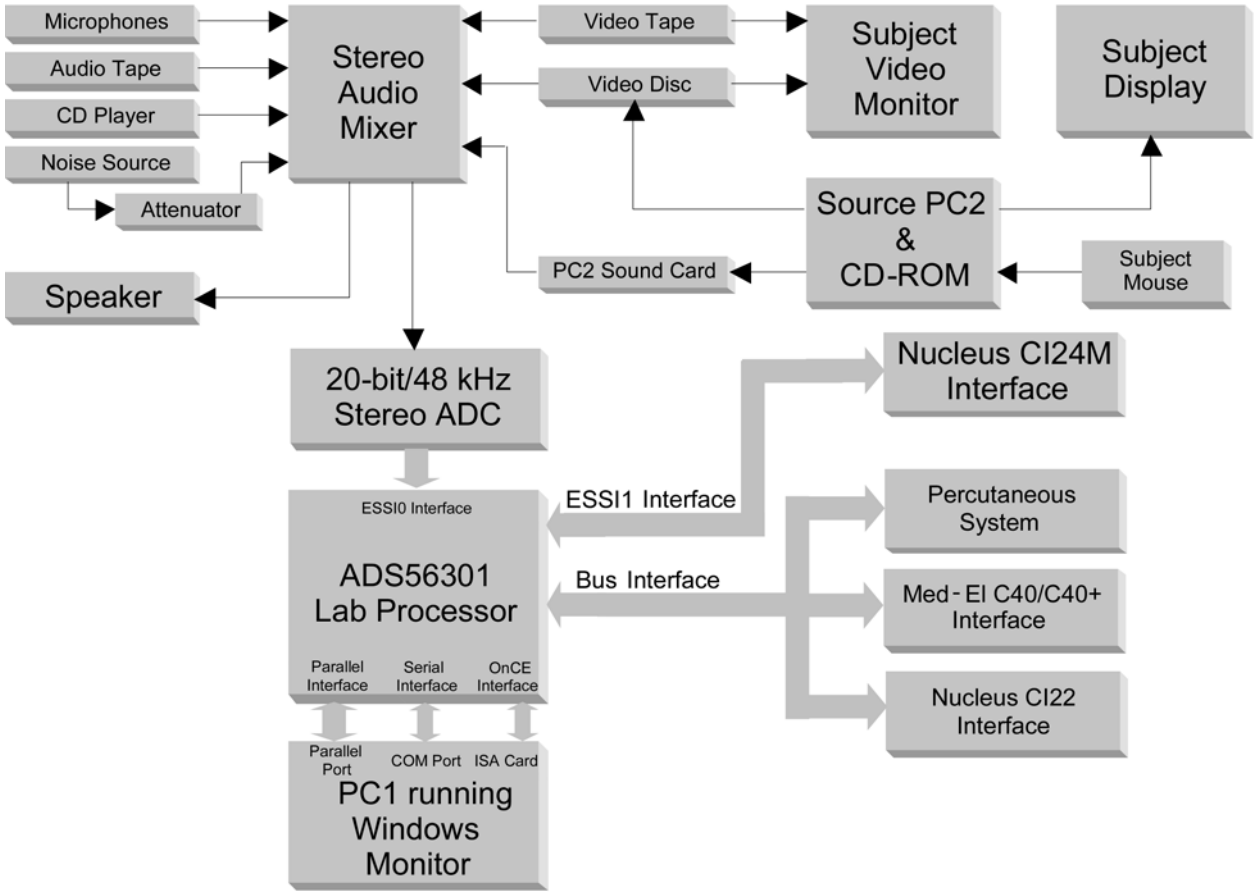


Figure 10: Block diagram of speech reception laboratory system with ADS56301 signal processor core

Interface to Nucleus CI22 Implants

The Nucleus CI22 implant provides sequential pulsatile stimulation through arbitrary bipolar pairs selected from 22 electrodes per ear. For safety reasons, the stimulation pulses must be biphasic with no residual current, *i.e.* the pulse widths of negative phase and positive phase should be equal. The CI22 implant interface receives pulse data through a parallel interface from the ADS56301 lab processor and generates amplitude modulated radio frequency bursts for power and data transmission to left and right inputs through a pair of inductive coils.

Our CI22 interface was previously connected to a dual DSP56001 laboratory processor controlled by DOS monitor programs. Recently, new DSP assembly programs were created to connect the CI22 interface to the more powerful DSP56301 laboratory processor, in order to take advantage of the extra processing power. The monitor programs of the CI22 interface were also upgraded to the Windows operating system to incorporate the CI22 implant into our unified user interface. Extensive verification and system tests of both the psychophysical and real-time speech processing mode have been completed for the newly developed system.

The CI22 implant uses an expanded or sync-electrode-mode-amplitude (SEMA) radio frequency data protocol, in which electrode, return electrode, amplitude, phase 1, and phase 2 information is encoded as radio frequency modulated bursts (see US patent 4,532,930). The maximum aggregate stimulation pulse

rate depends upon electrode configurations and amplitudes in addition to the pulse width. Using 40 μs/phase pulses, it can theoretically range from 2,685 pulses/s to 6,868 pulses/s.

The amplitude command can take the values from 0 to 239 clinical levels where the actual output current in the implant can be found in the output current table of each implant. The output current grows exponentially as the clinical level linearly increases from 0 to 239, which to some extent mimics the nonlinear loudness growth of normal hearing with respect to stimulation intensity. It is noteworthy that this nonlinear relationship between clinical level and output current makes it complicated to scale the stimulating current. Usually a look-up table is needed to find the clinical level for the linearly scaled current.

Because the radio frequency (RF) modulated signal is transmitted to the CI22 implant by an inductively coupled coil, the actual number of RF cycles received by the implant might not be the same as the number transmitted for a variety of reasons. Temporal redundancy is applied in the encoding of electrode and mode information such that the system is tolerant of up to ±4 RF cycles of variation in burst length. Amplitude and pulse widths are also affected by this transmission distortion. However, these variations are not as critical as those of electrode and mode, and there is no redundancy mechanism for amplitude and pulse width bursts.

Prior to the appearance of the first phase of a biphasic pulse in the implant, a sync burst, an electrode burst, a mode burst, an amplitude burst, and a pulse width burst have to be transmitted to the implant. A data value is encoded as the duration of each burst, *i.e.* the number of radio frequency cycles within each burst. The overhead of burst transmission time dramatically limits the overall stimulation rate. Moreover, the variation in electrode, mode and amplitude will change the actual timing of a pulse, making it extremely difficult to synchronize the pulses for bilateral implants. This difficulty is solved in the new DSP program for the CI22 implant interface, allowing synchronous control of bilateral CI22 receiver/stimulators.

It takes 6 bursts for the implant to generate one biphasic pulse: one sync burst and 5 data bursts (electrode, mode, amplitude, phase 1, and phase 2). The DSP has one timer for each implant to maintain the pulse rate by sending the first (sync) burst of each new pulse. The CI22 interface hardware relieves the laboratory DSP of the additional task of timing and counting the pulses in each command burst sent to the implanted circuits. The interface appears to the DSP as two separate memory spaces, one for each implant, which are loaded with the appropriate counts to generate the next control word bursts to each side. Within the interface, counters are loaded with those numbers and then decremented as the output pulses are generated. When a counter reaches zero, the interface generates an interrupt to the DSP, indicating readiness for the next burst count for that implant, and then begins to count down the minimum inter-burst interval. At the end of the interval, the interface will repeat this sequence if a new count has been loaded. Therefore, transmissions of data bursts are triggered by the interrupt requests from the interface. When a phase 2 burst is sent to the implant, the interface will send the 6th interrupt request to the DSP as well. If the DSP controls the pulse rate, it just ignores this interrupt request and waits for the next timer interrupt to send the first command burst of the next pulse to the interface. Otherwise, the DSP can send the first command burst of next pulse in response to the 6^{th} interrupt request from the interface, to achieve the maximum stimulation rate possible.

The CI22 implant expects a value within a certain range for each burst. For example, the sync burst should be between 1and 8 RF cycles, the amplitude burst between 16 and 255, and phase width bursts between 12 and 1,024. After receiving a valid burst, the implant will advance to the next state to wait for the next valid burst. If an invalid burst is received during any stage of a current pulse, the implant will abandon the current pulse and go to the initial state for a new pulse, *i.e.* waiting for the sync burst of the next pulse.

Comparison of CI22, CI24M and Med-El interfaces

The implant interfaces for Nucleus CI22 and CI24M implants, and Med-El's COMBI 40 and COMBI 40+ implants in our speech processor laboratory represent three different techniques of digital logic circuit. The way they communicate with the laboratory DSP is also different. The DSP programs that control these interfaces are designed to accommodate the requirements of each of these devices. Generally speaking, the more the interface can do, the simpler the DSP program. Before introducing the DSP programs for each implant, it may be helpful to present an overview of these interfaces.

The CI22 interface is built from discrete electronic components such as logic gates, registers, and resistors. It communicates with the DSP through a general-purpose parallel port. While the CI22 interface relieves the laboratory DSP of part of the burden of generating the RF transmitter signals, the DSP has to keep the timing for a desired pulse rate and provide as many as six command words in response to six external interrupt requests to generate each pulse in the implant.

The hardware of the CI24M interface is based on a Motorola DSP56303 EVM board with a clock rate of 80 MHz. Transmission of biphasic pulse definition data from the ADS56301 laboratory processor to the interface is accomplished through a synchronous serial connection. Two of the three synchronous transmitters of the second ESSI interface available on the DSP56303 chip (TX0 and TX1) are used to generate the modulation signal for the RF carrier. The modulated signals are directly fed to the implant transmitters. The DSP56303 on the EVM board maintains an internal biphasic pulse data buffer for each side. Whenever a buffer can be filled with new data, the EVM interrupts the ADS56301 DSP over the ESSI connection, requesting new pulse data for that side. Concurrently, the EVM continuously reads the data buffers to combine and output the implant modulation signals for both implants on TX0 and TX1. Because of the DSP56303 chip on board, the CI24M interface can accomplish much more complex tasks than the CI22 interface. The ADS56301 processor only needs to send two words of pulse data to the interface in response to the ESSI interrupt request.

The Med-El interface exploits a Field Programmable Gate Array (FPGA) technique to provide signals for its C40 and C40+ implants. The gate array connects to the DSP on the ADS56301 laboratory system through a parallel bus. The DSP controls the FPGA by writing to its registers. Separate registers are used to control stimulation to each ear. During startup, the DSP processor initializes the FPGA control and pulse data registers. Whenever the FPGA is ready to receive data for the next biphasic pulse, it generates an interrupt to the DSP (a dedicated interrupt is used for each side). In response to such an interrupt, the DSP writes three registers, which define electrode, current source reference, amplitude, pulse duration, and inter-pulse interval for a single biphasic pulse on one side. The FPGA takes these data to generate the appropriate command bursts for each transmitter.

Interface for percutaneous implants

Our interface to percutaneous electrode systems is designed for direct stimulation of implanted electrodes in subjects with percutaneous connectors. High pulse repetition rates (up to 10 kpulses/s/channel) and correspondingly short pulse widths (10-15 μs/phase) are supported. The interface consists of 24 independent signal DACs and current sources, which can be allocated to arbitrary combinations of 24 electrodes in one or both ears.

Compared with interfaces for Nucleus and Med-El implants, our current source interface can provide simultaneous stimulation to any combination of up to 24 electrodes, rather than only one electrode at any given time. Also, the timing of pulse phases in different electrodes are completely independent of each other in our interface, whereas other interfaces require that both phases of a biphasic pulse be finished

before a new biphasic pulse can be generated on any electrode. These features allow for investigation of various overlapping stimulation strategies and pulsatile strategies which utilize two or more intracochlear electrodes simultaneously.

One stimulation strategy that exploits the simultaneous stimulation feature of the interface is the use of conditioner pulses, which are high rate pulse trains with fixed amplitude. Conditioner pulses are generated simultaneously at a constant rate on all active electrodes, while stimulation pulses that are modulated in amplitude with the CIS channel output envelopes are interleaved between the conditioner pulses and across electrodes. The aim is to produce stochastic independence across the excited neural population, like that found in normal hearing but lacking in electrically stimulated hearing, at least using conventional stimuli.

Another stimulation strategy that exploits the capabilities of the percutaneous interface is virtual channel CIS (VCIS), where two or more adjacent electrodes are stimulated simultaneously to shift the perceived pitch with respect to the corresponding single-electrode percepts. Additional "virtual" channels can be produced in this way without increasing the number of electrodes.

Other novel stimulation strategies made possible by the percutaneous interface are under investigation and will be described in subsequent reports. Further hardware improvements are under way, involving the addition of a field-programmable gate array (FPGA) between the ADS56301 signal processor and the current source interface to enable higher stimulation rates, shorter pulse widths, a higher number of simultaneous stimulation channels, and/or to free additional DSP processing resources for complex speech processing algorithms.

DSP programs for bilateral psychophysics (CI22, CI24M, Med-El)

The DSP programs for CI22, CI24M and Med-El implants are similar in structure. They all consist of three parts: the diagnosis mode, psychophysics mode, and speech-processor mode. Within each mode, the program flows of the three implants are also similar. We shall use the CI22 implant as an example to describe the DSP programs used with all three implants.

The monitor PC downloads the DSP program through an ISA slot to the on-chip emulator (OnCE) port of the DSP. When the CI22 DSP program starts, it enters a diagnosis mode where the operator can control the flow of the program or request hardware/software information from the DSP. In the diagnosis mode, the user can choose to go to the debug mode, reset the DSP, check the processor identification or software version, download data from PC to DSP or upload data from DSP to PC. These control commands and download/upload data streams are transmitted through the serial connection between the COM port of the PC and the serial communication interface (SCI) port of the DSP. The DSP program keeps polling the SCI port for new instruction when it is idle and immediately sends back any received bytes to the monitor PC. The user can choose to enter a psychophysics or speech-processor stimulation mode from the diagnosis mode using dedicated SCI commands. When the user exits the psychophysics mode or the speech-processor mode, the DSP program returns to the diagnosis mode.

Upon entering the psychophysical mode, the DSP program first clears both left/right background and pulse data tables to ensure that no random data will be sent to the implant, sets priority levels for core and peripheral interrupts, and then goes to a foreground loop of polling the SCI port for commands and data from the monitor PC. The user can set up pulse data tables in the DSP X/Y memories and also control the configurations of left/right side stimulation from the monitor PC. The user can also start or stop the delivery of stimulation pulses through SCI commands, as specified from the monitor PC. To keep the

implant powered, background pulses with minimum amplitude are sent when no explicit stimulation is taking place.

When the DSP program receives the command to start a pulse sequence, it enables and sets Timer0/Timer1 in the DSP according to the pulse rates for the left/right side, so that a timer interrupt occurs at the moment that a new pulse should be sent to either side. The DSP sends a sync command to the interface in the timer interrupt service routine to initiate a sequence of IRQA/IRQB interrupt requests from the interface for the left/right side. In response to the first interrupt request from the interface, the DSP sends the electrode command to the interface in the IRQA/IRQB interrupt service routine. The interface generates a 2.5MHz radio frequency modulated burst whose width is determined by the value of the electrode command. This burst is sent to the implant through a transmitter coil. When the electrode burst is completed, the interface sends another interrupt request to the DSP to ask for a mode command. In this way, the interface sequentially obtains the remaining amplitude, phase width 1, and phase width 2 commands from the DSP for each stimulus pulse.

In the psychophysical mode, sequences of pulses are sent to the implant to obtain measures of the threshold, most comfortable loudness level (MCL), tone perception of each electrode, interaural timing differences (ITDs), and interaural amplitude differences (IADs) using selected bilateral pairs of electrodes. Usually the patterns of stimulation pulses used in psychophysical testing have less amplitude and frequency variation than is present in processed speech signals. A pulse data table is generated by the operator in the monitor PC which includes all the information that is needed to make each pulse for a given stimulation pattern. In this pulse data table, each command of electrode, return electrode, amplitude, or pulse width is a 24-bit word. In addition to the pulse information words, several command words are also available for use in the pulse data table, such as a repeat command to make periodic pulses, a synchronization command to synchronize pulses between the two sides, and an end-of-table command to terminate the current stimulation pattern. Arbitrary delays can be assigned to either side to produce synchronized pulses.

Synchronization between sides of bilateral implants is critical for measurements of sensitivity to interaural timing or amplitude differences. The CI22 interface does not provide the function of synchronization. Instead, the DSP program for the CI22 implant has to synchronize the pulses to both sides in the DSP timer interrupt service routine if a synchronization command is encountered in the pulse data table. The synchronization of CI22 implants is complicated even more by the fact that the actual timing of a pulse after the pulse start command (sync burst) is a function of its electrode address, mode and amplitude. To synchronize the timing of pulses to both sides, the DSP program needs to make adjustments to the pulse timing with respect to its electrode, mode, and amplitude variation while still maintaining the prescribed pulse rate.

The interfaces for CI24M and Med-El implants are more sophisticated than the CI22 interface in that the interfaces are capable of providing synchronized pulse data for bilateral implants. When the interfaces receive a synchronization command from one side, they will wait for the synchronization command from the other side before they send pulse data to the implants. The Med-El interface can be used for COMBI 40+ implants, COMBI 40 implants, and even a combination of these two implants. For mixed implants, synchronization becomes complicated even with the synchronization function of the interface because of the different clock rates of the two implants. When the interface sends pulse data to both implants at the same time, the faster COMBI 40+ receives the data earlier and starts a pulse earlier than the COMBI 40. Alignment pulses, with zero amplitude and with different durations, are sent before the actual pulse data to offset such time differences, resulting in simultaneous pulses at both sides.

DSP programs for bilateral speech processors (CI22, CI24M, Med-El)

Unlike the psychophysics mode, in which pre-determined pulse patterns are sent to the interfaces, the speech-processor mode receives real-time speech samples from an analog-to-digital converter (ADC), processes these digitized samples in real-time according to a given signal processing strategy, like the continuous interleaved sampling strategy (CIS), and sends the strategy output to electrodes in a chosen stimulation order. Such differences in functionality make the DSP program structure of the speech-processor mode fundamentally different from that of the psychophysical mode. The foreground loop of the DSP program keeps checking if a new speech sample has arrived from the ADC. When a new sample arrives, the DSP program starts the CIS filtering and other operations.

The ADC alternately transfers left and right samples to the DSP through an enhanced synchronous serial interface (ESSI 0). Each sample includes one word containing the 20-bit sample value and one word identifying whether the sample is from the left or right side. Each ADC transmission results in 2 words being written to DSP memory: One word contains the 20-bit sample value and a second word identifies whether the sample is from the left or right side. Availability of the left/right word for each ADC sample allows the DSP to detect any failure of processing to keep up with the output pulse rate and, in that event, shut down in an orderly way.

The sampling rate of the ADC is 48 kHz per side, but the filters for the CIS processing are designed to run at 24 kHz, so the speech samples are first down-sampled from 48 kHz to 24 kHz. For both left and right sides, two input samples are down-sampled to one by using anti-aliasing filters. The 24 kHz samples go through a pre-emphasis equalization high-pass filter before being processed through a bank of band-pass filters, followed by a rectification/low-pass filter envelope extraction stage and an instantaneous maplaw compression stage.

Several types of interrupts with different priority levels are used in the speech-processor mode. ADC samples are sent to DSP memory by an ESSI 0 data-receive interrupt. In order not to miss any input samples, this interrupt has the highest priority level of 2. With the CI22 implant interface, Timer0/Timer1 interrupts with a priority level of 1 occur at the pulse rate on the left/right side to start generating a new pulse. IRQA/IRQB interrupts with priority level 1 are used by the interface to request pulse data from the DSP. The monitor PC communicates with the DSP through a SCI data-receive interrupt of priority level 0. This interrupt has the lowest priority level in order not to interfere with receiving input samples and sending pulse data to the interface.

After CIS filtering, the channel output envelope signals are sampled to a stimulation amplitude vector with each element corresponding to one distinct frequency band and channel. Each channel is usually assigned to one electrode. Rather than all active electrodes being stimulated simultaneously, they are usually stimulated sequentially in a specified order. If necessary, some electrodes can even be stimulated more than once during each stimulation cycle. To accomplish these assignments, a channel data table is used for each side. For the CI22 implant, the channel data table for each stimulation pulse consists of a group of 7-word blocks ordered according to the stimulation order of the channels. The 7 words in each block are electrode, mode, pointer to CIS filter output, pointer to maplaw table, pointer to channel-specific volume factor, pulse width 1, and pulse width 2 of a channel.

Monitor program task for single-burst psychophysics

The monitor program runs in a Windows 98 environment, enabling us to use the Motorola library functions for accessing the DSP through the Motorola OnCE interface. The program's design is based on object-oriented programming techniques and it is implemented in C++ in the Borland C++ Builder 5

development environment. The monitor is a menu driven graphical interface that allows the operator to perform a variety of tasks and procedures for controlling the DSP and any of its interfaces. The DSP needs slightly different data for each of the different interface systems. The monitor has separate classes for each of these different systems, but the classes' interface to the rest of the program is the same in each case. Taking advantage of object programming, all the monitor's different task programs can be used with all the interfaces, so long as the implant being interfaced can support the particular mode of stimulation involved.

The psychophysics program

This program enables operators to send pulse trains and increase or decrease the stimulus amplitude to obtain threshold, comfortable levels, or other levels. It can be used unilaterally or bilaterally but can only generate one pulse train at a time. This program is generally used to acquire threshold and MCL levels for speech processors and/or other psychophysical tests. It can also be used for informal pitch discrimination, scaling, and pitch-matching tests, allowing stimulation with any of a set of different pulse trains pre-defined in a table. Up to 44 separate pulse trains can be defined, allowing for all electrodes in a bilateral CI24M patient to be stimulated in one session.

Different kinds of pulse-train definitions are possible:

- Bi-phasic, charge-balanced pulses.
- Tri-phasic, charge-balanced pulses. (Percutaneous systems only)
- Sinusoidally modulated pulse trains, with the operator defining the modulation rate and depth of modulation in the table.
- Pulse trains comprised of pulses with different pulse widths. (An example is shown in Figure 11, where token 1 will have 3 different pulse widths. The first pulse will have a pulse width of 30 μs/phase, followed by a pulse with 40 μs/phase and one with 100 μs/phase and then the cycle is repeated. The period is defined from the middle of one pulse -- at the change of phase -- to the middle of the next pulse. The shortest pulse receives the amplitude defined in the table while the others receive amplitude inversely proportional to their pulse widths to maintain constant charge across pulses.)
- Pulse trains with biasing pulses in the background. (The biasing pulses or conditioners are defined separately and run at a different rate than the foreground pulses. The conditioners can be multiple electrodes running simultaneously or sequentially at a high rate. The amplitude of each conditioner electrode is specified separately, as illustrated in Figure 12. Once conditioners are enabled in the psychophysics tests they run continuously in the background until the operator stops conditioner stimulation.) (Percutaneous systems only)
- Virtual channels where more than one electrode is stimulated to produce a pitch percept different from that obtained with any of the electrodes alone. (Figure 13 shows a separate pop up window which appears to help the operator enter the VCIS channel specifications. Up to 6 electrodes can be defined for monopolar stimulation with a corresponding multiplier. In the example shown, the operator is trying to produce a lower pitch percept than stimulating electrode 1 alone.) (Percutaneous systems only)

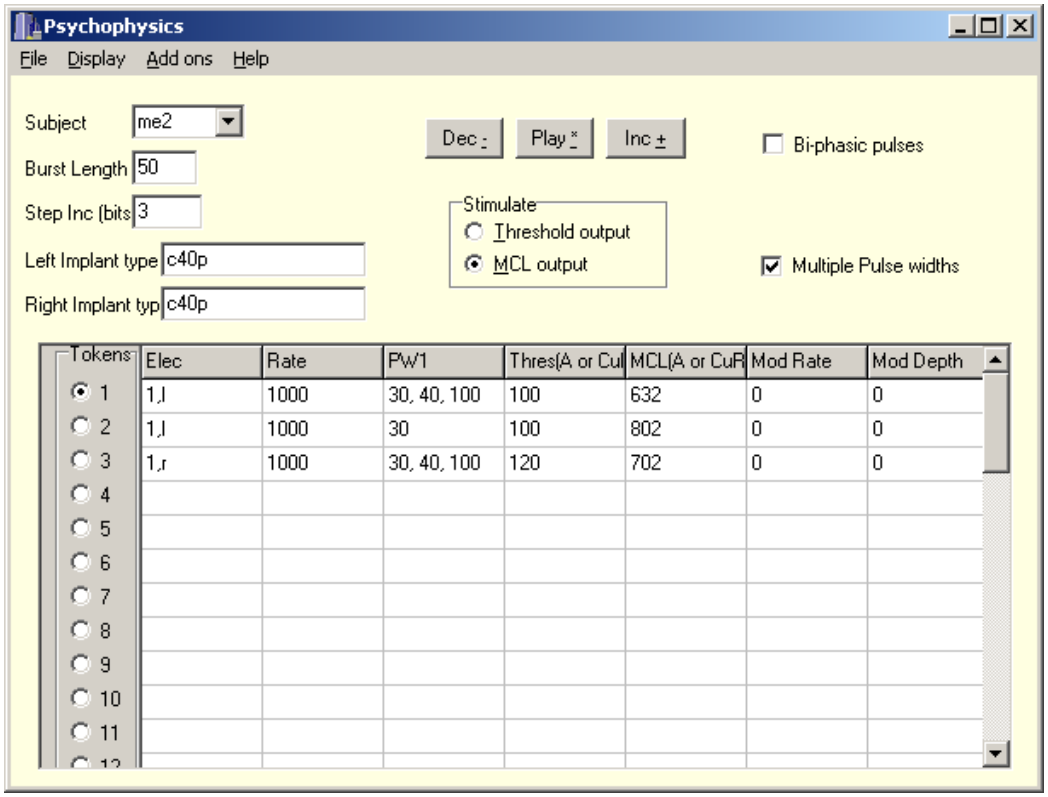


Figure 11: Psychophysics form displayed for Med-El interface

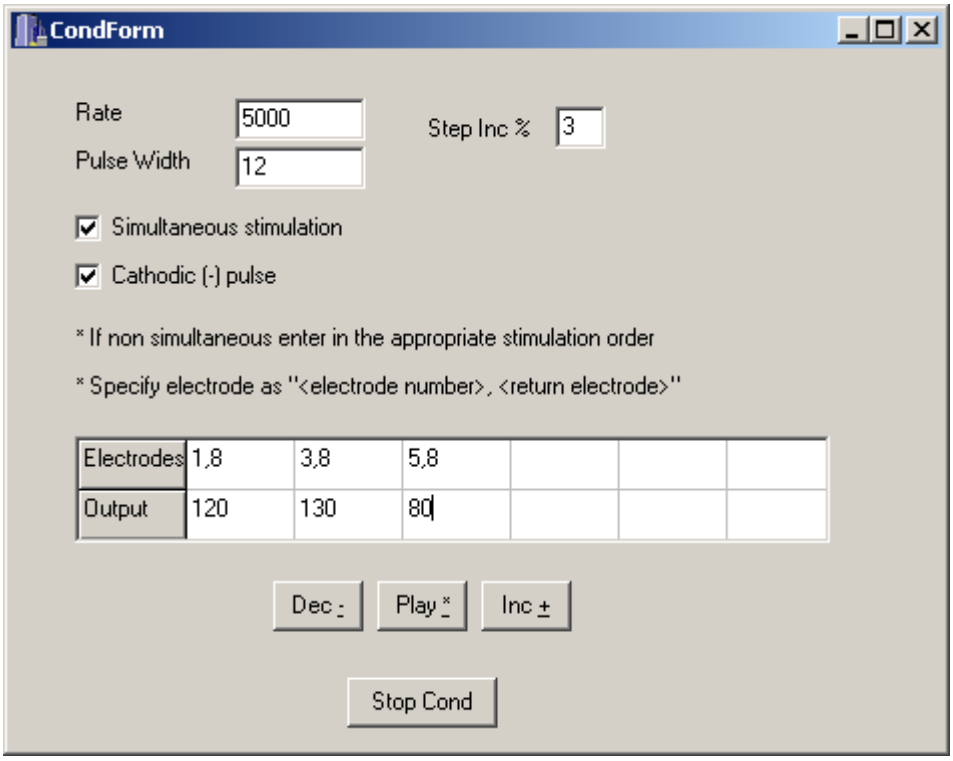


Figure 12: Conditioner pop-up window used to define fast conditioner-pulse stimulation for psychophysics tests

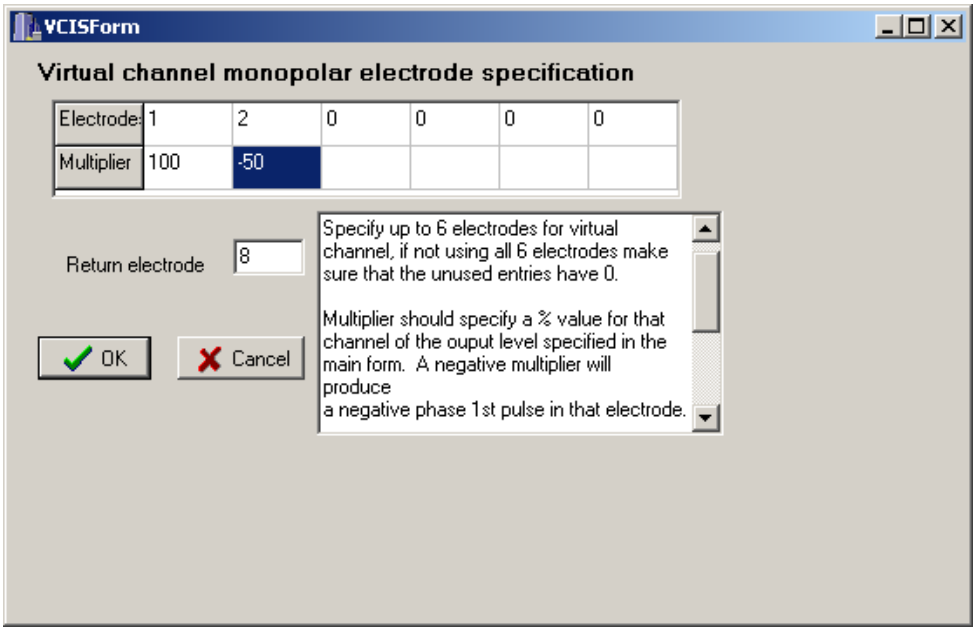


Figure 13: Virtual-channel pop-up window to help the operator specify virtual channels in psychophysics tests

The operator sets up the pulse train specifications in the table, chooses the active channel with the corresponding radio button, and then presents the stimuli using the play, increment, or decrement buttons.

The pulse trains defined in the psychophysics program table, human subject identification code, and implant type specification can be saved into a text file. This file can be used to help generate speech processor spec files, to reload the information to the psychophysics form at a later time, or as a basis for running other psychophysics tasks, such as scaling and ranking tests, as indicated below.

Other psychophysical tasks

Other tools that run a variety of psychophysical tests in the monitor include:

- Scaling test - can be used for pitch scaling, loudness scaling, or discrimination tests. It uses a text file generated by the psychophysics test from its stimulus table to present one pulse train at a time in a random order. After the subject hears a stimulus train, (s)he enters a number describing the sound. According to the instructions given, this number can be anywhere between 0 and 100, specifying a pitch level, scaling factor, or loudness.

- Ranking test - similar to the scaling test but with two or more bursts presented sequentially and the subject choosing which sound was higher in pitch, louder, or different. The ranking test can be used for pitch or loudness ranking according to the instructions and set-up of the test.

- Forward masking test - uses a four alternative forced choice paradigm to obtain a threshold using a masker and probe or the probe alone. Both the probe and masker can be an unmodulated pulse train, a modulated pulse train, or a single pulse. The stimuli can be delivered to a single monopolar electrode, or to electrodes in a variety of other configurations, including “virtual channel” configurations.

Monitor program task for bilateral psychophysics

Lateralization tests

The lateralization program is used to measure the bilateral subject’s ability to lateralize on the basis of some difference between pulse trains delivered to the two sides. The two pulse trains are specified by the operator in a table on the form (see Figure 14) and can be sinusoidally modulated bursts or simple pulse trains. The operator must calibrate the pulse train amplitudes before starting a test, by stimulating the two electrodes simultaneously and adjusting their amplitudes until the subject perceives the sound at midline.

The lateralization form is shown in Figure 14. In this example we have a subject with bilateral Med-El implants. The pulse trains are defined in the table with the following parameters:

- Electrode and side – to run a lateralization test the sides must be different.

- Rate – normally this is the same rate for both electrodes, but the operator has the ability to specify any rate possible with the implant and pulse width specified.

- Pulse width

- Output amplitude specified in either µA or clinical units.
- Modulation rate and depth of modulation can be defined for sinusoidally modulated bursts or set to 0 for simple pulse trains. Modulated bursts will start at phase angle 0 unless otherwise specified in the phase page of the Lateralization Test Type box.
- Burst length is the length of one pulse train burst defined in ms. During one presentation the pulse train is generated 3 times with a delay between bursts in ms, as specified in the Time Between Burst edit box. Figure 15 shows the 3 presentations of a 300 ms burst with a delay of 500 ms between the bursts.

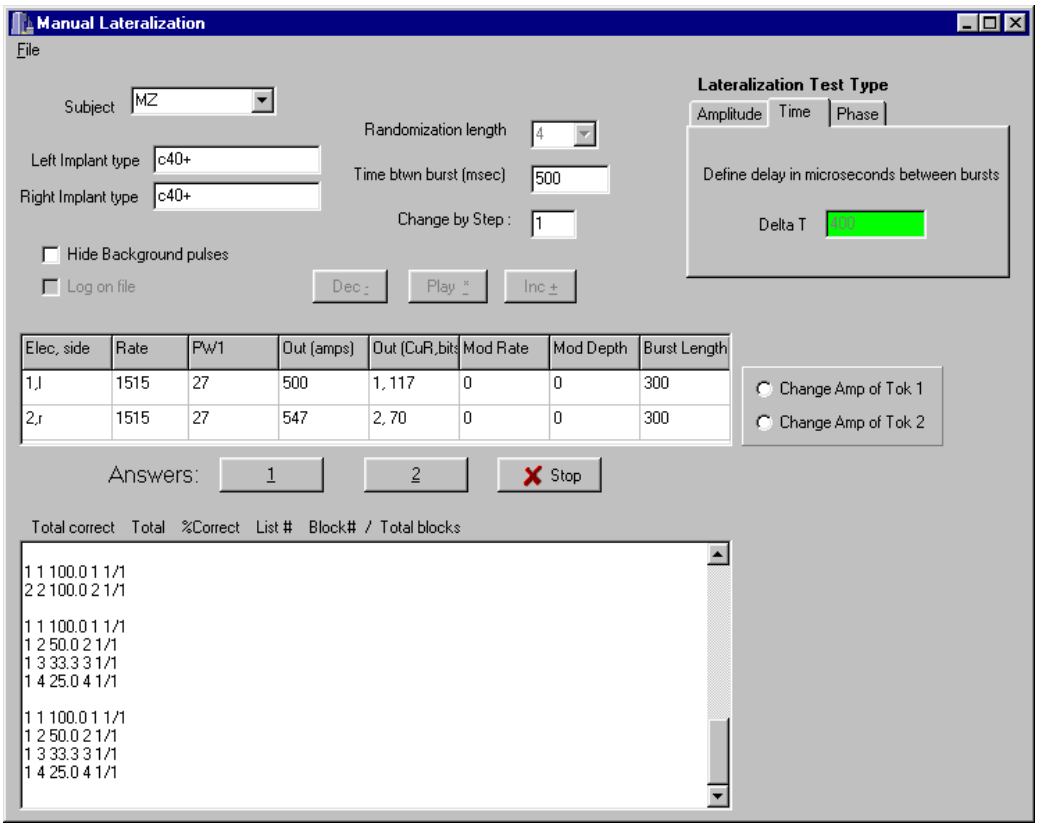


Figure 14: Lateralization test form set up for ITD test on a bilateral Med-El subject

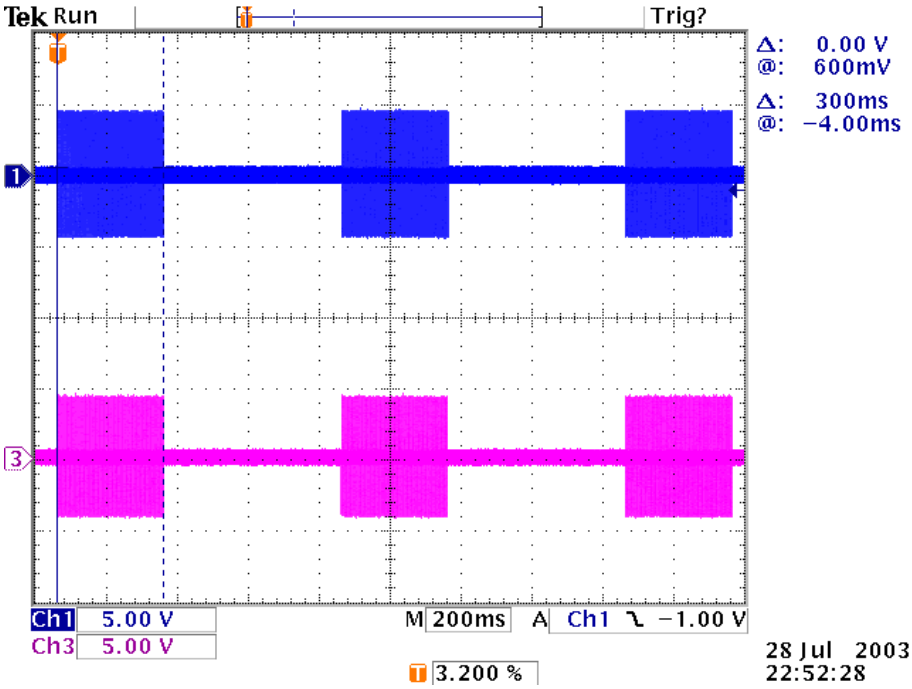


Figure 15: Oscilloscope traces of the full 3 bursts in an ITD test as specified in Figure 14. The upper trace corresponds to the left side and the lower trace to the right.

Three different tests can be run from the Lateralization form, they include:

- Interaural timing difference (ITD)
- Interaural amplitude difference (IAD)
- Interaural modulation phase difference (IPD)

Interaural timing difference tests

The operator chooses the number of presentations in a subtest in the Randomization Length list box and defines the time delay between the start of the two bursts in µs. In the example shown in Figure 14, the 4 presentations are randomized for one subtest run with a 400 µs delay between the start of the first and second bursts on the two sides. Figure 16 shows in detail the beginning of the pulse train; the same delay will appear in all three bursts presented. The subject then reports whether the sound is more to the left or to the right. The operator records the answer by pressing button one or two, corresponding to the first (usually left) or second (usually right) token definition in the table. The total number of tests, total number of correct answers, percent correct, and position in the test are displayed in the Memo box and logged to a file if enabled on the form. Double clicking the DeltaT box will highlight that entry box. Once this is done the Increment and Decrement buttons (as well as the +/- keys on the keyboard) will increment or decrement the time delay by the specified step on the form. The operator can now easily change the time delay and continue the tests. If the delay or any other burst specification is changed, the percent-correct score is reset to 0; otherwise it continues to generate a value using all the previous test data.

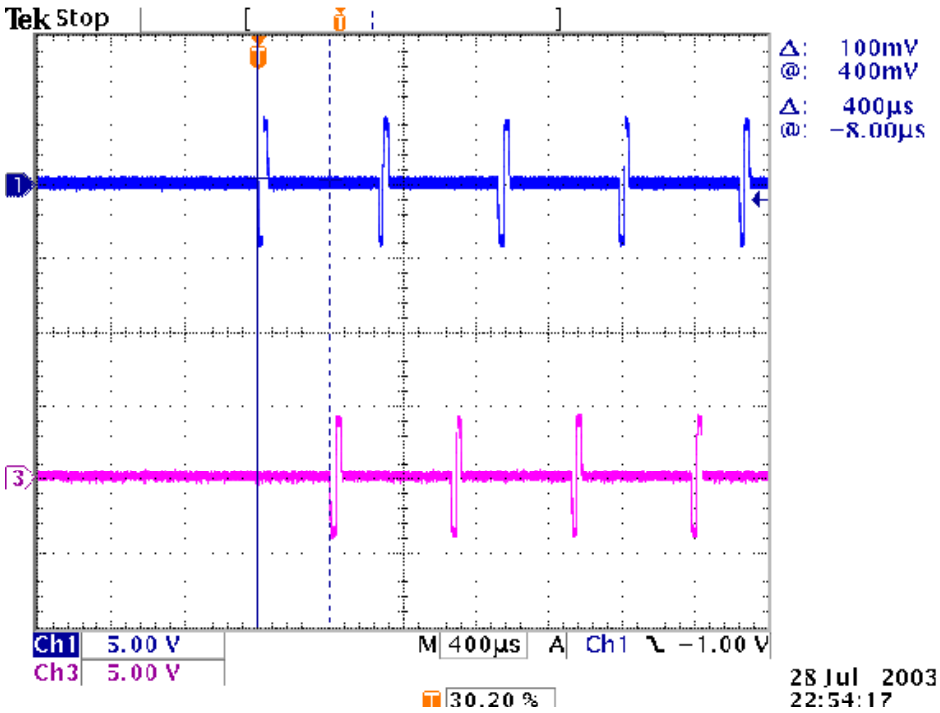


Figure 16: Oscilloscope traces of beginning of trains shown in Figure 15 at higher time resolution. The right side stimulation (lower trace) begins 400 µs after the left side stimulation (upper trace).

Interaural amplitude difference tests

Interaural amplitude difference tests may be run manually, with the operator changing the amplitude of one pulse train definition at a time. No randomization list is used; the operator manually changes the levels up and down being careful always to leave one electrode's amplitude set equal to its reference amplitude level. The levels presented are compared to reference levels specified for each pulse train (these are set to amplitude levels that together produce a midline percept) to determine whether the subject's lateralization response is correct. The operator then manually changes the amplitude of either electrode with the +/- buttons and plays the new stimulus. Output amplitude difference with respect to reference levels is normally started around 4 clinical units and decreased until the subject cannot reliably report the side with an amplitude advantage. (Separate software also is available to measure sensitivities to IADs using automated procedures, see e.g. QPR 1, NIH project N01-DC-8-2105, 1998.)

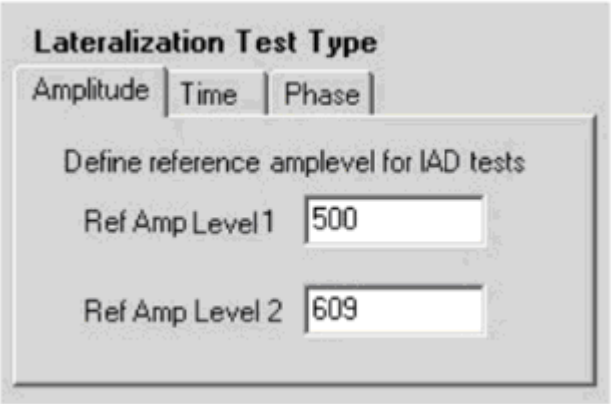


Figure 17: IAD test window, showing the tab where operator specifies reference amplitudes

Interaural modulation phase difference tests

For phase difference tests, the operator defines 2 modulated pulse trains and balances the amplitudes to obtain a midline percept when both are presented simultaneously and starting at the same phase. The operator defines a phase shift between the tokens in degrees. Tok1 and Tok2 referenced in the lateralization phase tab (see Figure 18) do not refer to the electrodes defined in the table but to the "first token" and to the "second token" as defined in a randomization list (this test uses the same randomization lists as the time difference test). Thus phase shifts become randomized between the two pulse train definitions. The subject reports whether the sound is more to the left or to the right. The correct answer should be the side corresponding to the token with the leading phase. Figure 19 shows a close-up of the starting phases for the test specified in Figure 18. Because the upper trace corresponding to the left side is lagging by 90 degrees in this example, the subject should hear the sound as coming more from the right. Once again the monitor shows the percent correct and total number of runs. Once a randomization set is completed, the operator changes the phase shift if desired and continues until the subject cannot reliably report the side with leading phase.

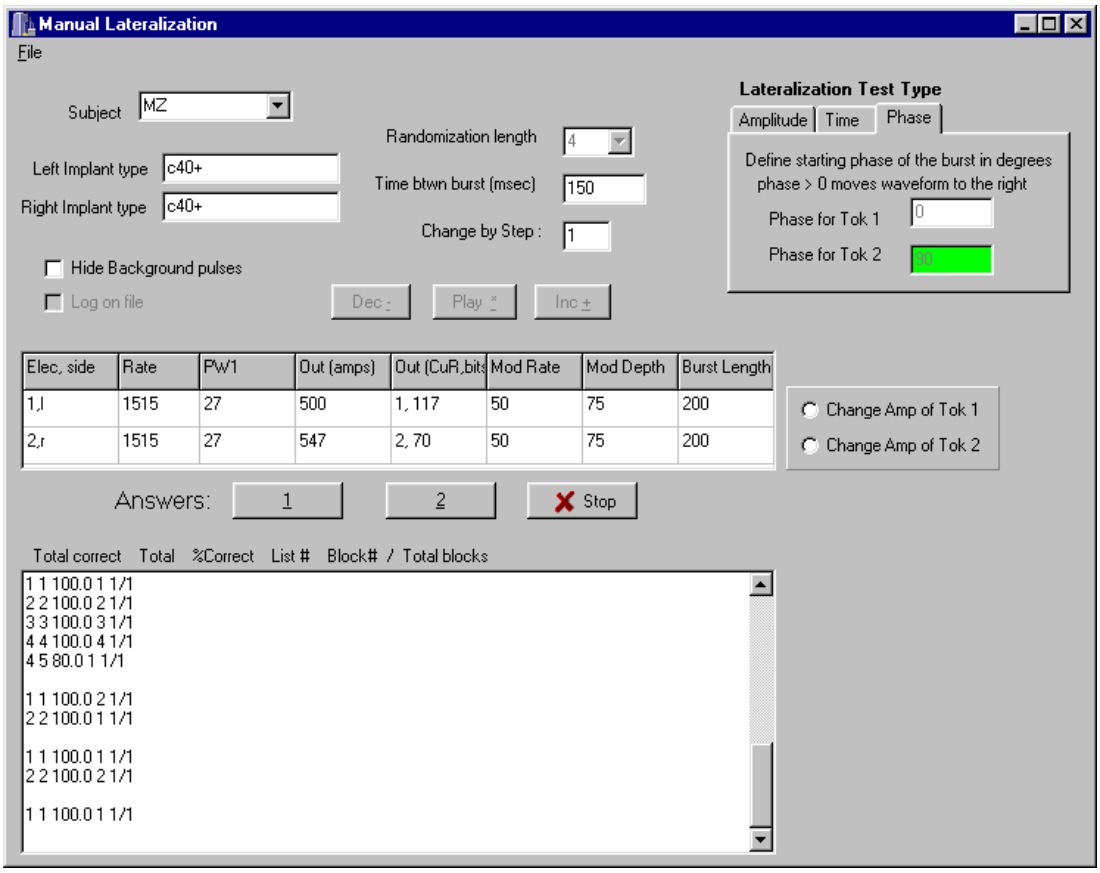


Figure 18: Lateralization form set-up for phase difference tests

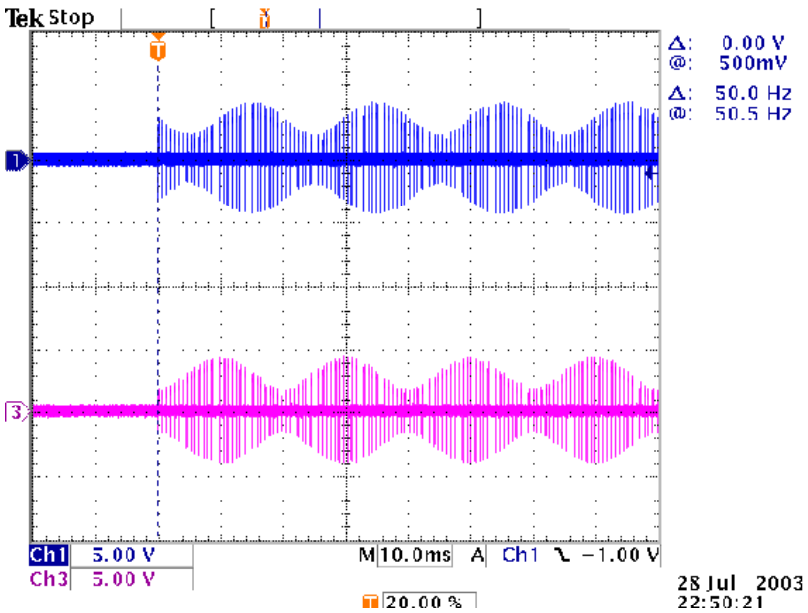


Figure 19: Oscilloscope traces of the start of the pulse trains for the phase difference test specified in Figure 18. The upper trace shows the left side, the lower trace the right side, and the relative phase delay (randomly applied to either side) is 90 degrees and has been applied to the left side here.

Setup and control of unilateral and bilateral speech processors in the monitor

The speech processor loading tool in the monitor program downloads unilateral and bilateral processors in the same manner. The download procedure is similar for all interfaces. The operator opens a speech processor specification file, which is displayed in the bottom memo pad of the form. If needed, the spec file can be modified within the memo pad and the modified version saved as a new spec file.

The speech processor spec file is the heart of the process, as it defines the DSP speech processor program, parameter variables, and all patient-specific data needed for the monitor to download and run the processor. Figure 20 is a sample spec file for a bilateral Med-El 8-channel processor. The lines starting with an asterisk are comments and are ignored by the monitor. Any line starting with greater than sign is a key word that the monitor looks for. If a particular key word is not found a default value is used, if one exists, or an error message will appear to inform the operator of the missing data. The spec files contain the following information:

- Implant type identification – if bilateral, both sides need to be specified.
- ADC input mode – mono or stereo. If not specified, mono is assumed.
- Load files – DSP program/filter coefficient files.
- Channel definition - this includes electrode, threshold, MCL value, and pulse width in µs. (The data format varies among interfaces. For a bilateral processor, channels must be specified for both sides.)
- Rate of stimulation - for a bilateral processor rates must be specified for each side.
- Rectification method – full-wave or half-wave rectification for each side. If not specified, full-wave is assumed.
- Channel stimulation order – order must be defined separately for each side.
- Maplaw input gain – if not specified, a gain of 0x1ffe is used.
- Maplaw exponential – Either a single exponential to be used for all channels or a separate exponential for each channel. If bilateral, both sides must be defined.
- Conditioner information – only for percutaneous systems. Provide the conditioner specifications including electrodes, rate, pulse widths, and output levels.
- Mode - for bilateral processors only. Specifies whether the processors are to run synchronously or independently. Synchronous processors can be specified further to be synchronized at the start of each channel order cycle (allowing synchronization of two processors with different numbers of channels), or to be synchronized at every pulse. If synchronous, a delay for either side in µs can also be specified.

Figure 20, an example of a specification file for an 8-channel bilateral speech processor, is presented on the following pages.

```
* CT6             6/11/2003 3:22PM
*
* System:         ADS56301 Lab system with Windows ADSMonitor
* Interface:      Bilateral MED-EL XILINX interface
* Implants:       C40P left and right
* CIS processor:  stereo input, bilateral, synchronous 8/8 channel processor
*                 BP filters 1-8 assigned to electrodes 1,3,5,6,7,8,9,11 left
*                 BP filters 1-8 assigned to electrodes 1,2,4,5,6,7,8,10 right
* EQ filter:      1st order HP at 1.2 kHz cutoff
* BP filters:     6th order, log. spacing in range 350 - 5500 Hz left
*                 6th order, log. spacing in range 350 - 5500 Hz right
* Env. extractor: full-wave rectifier and 4th order LP at 200 Hz cutoff
* Stimul. params: 1515 pps per electrode left,
*                 1515 pps per electrode right,
*                 staggered stimulation order
*
>system
MedEl
>implant left
c40p
>implant right
c40p
* ADC input mode
>input
stereo
* DSP program and filter configuration CLD files
>load files
d:\dsp563\medel\ADS1XIL.cld
d:\dsp563\Filtersets\ADSbilateral\S88N2004.cld
*Channel definitions - eles  current range  thres  mcl  pw[us]
>channels left
1 2 8 92 26.7
3 2 7 87 26.7
5 1 16 124 26.7
6 1 13 125 26.7
7 2 6 69 26.7
8 2 6 70 26.7
9 2 7 69 26.7
11 1 19 112 26.7
>channels right
1 2 7 112 26.7
2 2 7 109 26.7
3 2 8 100 26.7
5 2 9 90 26.7
6 2 7 85 26.7
7 2 9 79 26.7
```

```
8 2 9 77 26.7
10 2 10 76 26.7
* Rate in pulses per second
>rate left
1515
>rate right
1515
* Rectification in envelope extractor:
* use full-wave (default)
*>rect left
*half
*>rect right
*half
* Channel order:
* Numbers here designate channel numbers as defined in the
* channel definition tables, not electrode numbers
>order left
1
5
2
6
3
7
4
8
>order right
1
5
2
6
3
7
4
8
* Maplaw input gain
* Commented out, so use default value 8191 = 0x1FFF
* >gain left
* 9010
* >gain right
* 9010
* Map exponents
>map left
-0.0001
>map right
-0.0001
* Synchronization settings:
```

```
* synchronize pulses on both sides, no delay
>mode
sync
```

Figure 20: Example of a specification file for an 8-channel bilateral speech processor

The monitor uses the information in the spec file to generate the appropriate channel tables, initialize variables, and generate the compression tables for the DSP. There are times when the rate specified in the specification file cannot be attained. This might be due to large pulse width definitions or the rate may have to be modified in order to synchronize both sides. The monitor will inform the operator in a pop-up window of the closest rate it was able to implement. The implemented rate, any default values used, and other status information is displayed in the status memo of the form.

The monitor generates an individual compression table or maplaw for each of the channels. It uses the threshold (THR) and MCL values as well as the maplaw exponential defined in the spec file to calculate a table with 1024 entries.

The equation used for the compression tables is:

$$a = \frac{MCL - THR}{1024^{\exp} - 1} \qquad k = MCL - a * 1024^{\exp}$$

$$out(n) = a * n^{\exp} + k$$

where "exp" is the power exponential defined in the specification file and is normally -.0001 (to produce a close approximation to logarithmic mapping). 1024 is the total number of entries in the table and n is the table index from 1 to 1024.

Pressing the Go button sends a command to the DSP to start the speech processor. In the transcutaneous systems the volume of the speech processor is controlled by software. One or two volume control boxes will appear in the form depending on whether the speech processor is unilateral or bilateral. For percutaneous systems, the volume is controlled by hardware in the operator's control box. For a speech processor with conditioner pulses in the background, the software controls the amplitude of the conditioner and a similar volume box as seen in the form in Figure 21 will appear. The stop button will send a command to the DSP to stop processing in an orderly way, after which a new spec file can be loaded.

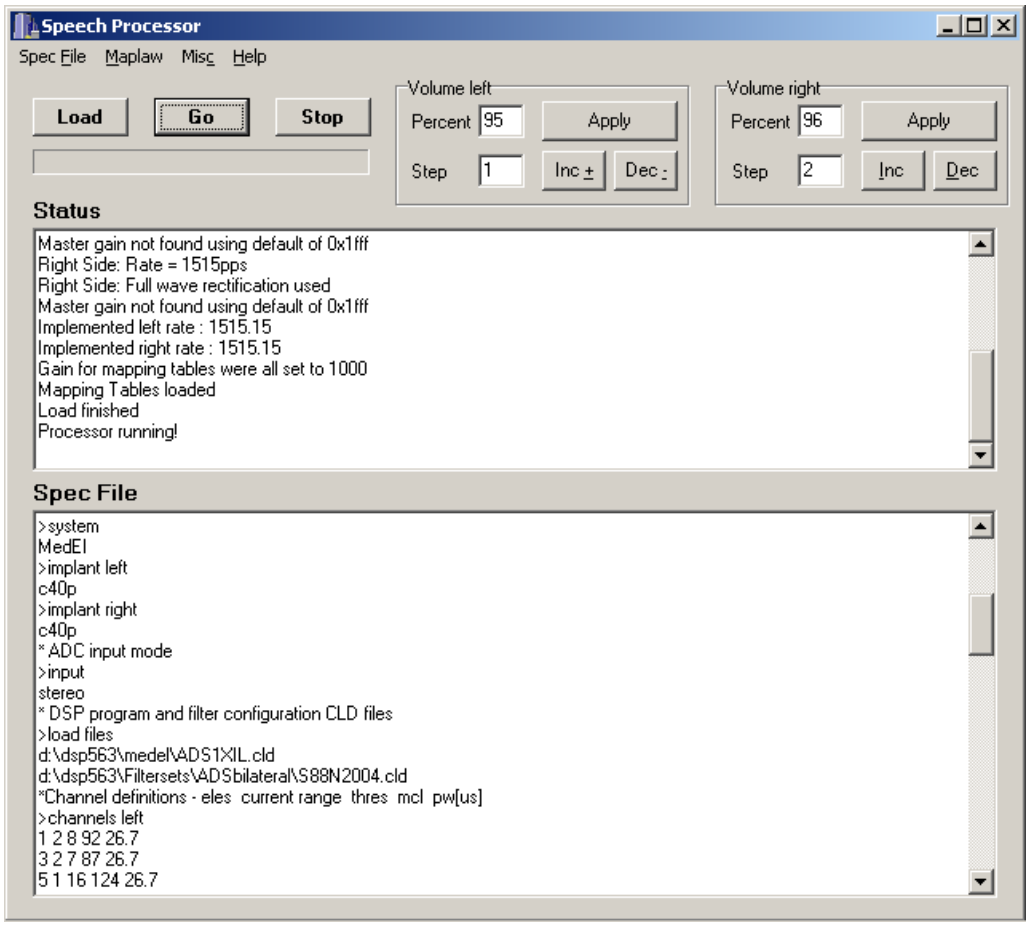


Figure 21: Speech processor download form

Appendix 3: Source code listing

MakeAMP.m

```
function stratpar = MakeAMP(varargin)
% stratpar = MakeAMP( ... )
%
% Processes wave files, according to specified strategy, taking speech
% processor parameters from specified ADSMonitor spec file and from either
% the command line or a strategy parameter structure function argument, if
% supplied. The resulting output files are amplitude sequence files for
% use with the ADS56301 streaming mode.
%
% Processing parameters scanned from the spec file:
%   - Implant interface system
%   - Number of channels
%   - Pulse rate per channel
%   - Stimulation order of electrodes
%   - VCIS channel information (if present)
%
% All remaining signal processing parameters are entered from the
% command line or supplied via parameter structure argument (see below).
%
% Function calling syntax:
%
% MakeAMP( [strategy[, specfile[, wavfile[, side]]]] )
%
%   strategy  String or cell array containing 1 (for unilateral spec files) or 2 (for bilateral
%             spec files) of the following signal processing strategy strings:
%             'stdCIS'      Standard CIS algorithm (Butter BP filter bank,
%                           LP/rectifier as envelope extractor)
%             'cpCIS'       CIS with bank of nonlinear DRNL filters and
%                           LP/rectifier as envelope extractor)
%             'stdihcCIS'   CIS with Meddis IHC model as envelope extractor
%             'stdihclpCIS' CIS with Meddis IHC model/LP filter as envelope extractor
%             'stdhpCIS'    CIS with adaptation HP stage after envelope extractor
%             'cpihcCIS'    CIS with bank of nonlinear DRNL filters and
%                           Meddis IHC model as envelope extractor
%             'cpihclpCIS'  CIS with bank of nonlinear DRNL filters and
%                           Meddis IHC model/LP filter as envelope extractor
%             The selected signal processing strategy will be applied to both
%             left and right side for bilateral processors.
%             If omitted, the processing strategy is interactively enquired
%             from the command line. For bilateral processors, 2 different
%             strategies can be selected for left and right side in this case.
%             Signal processing parameters are the also enquired interactively
%             from the command line.
%
%   specfile  ADSMonitor speech processor spec file containing parameters
%             like number of channels, stim. order, stim. rate. If
%             omitted, file requester is opened.
%
%   wavfile   Name of Microsoft wave sound file. If omitted, file requester
%             for selecting 1 or more wave files is opened.
%
%   side      Side which is HRTF processed with stereo input wave files ('left' or 'right').
%             If not specified and stereo file is encountered, a message box is
%             openend to ask for side.
%
% MakeAMP( strategy, stratpar[,specfile[, wavfile[, side]]] )
%
%   strategy  Same as above
%
%   stratpar  Cell array containing 1 (for unilateral spec files) or 2 (for bilateral
%             spec files) parameter structures. A parameter structure contains some or
%             all parameters relevant to signal processing strategy (see below for
%             definition of parameter structure). Structure elements not defined are set
%             to default. If argument is omitted, parameters not in [ ] relevant for the
%             selected strategy must be interactively entered from the command line.
%
%     Field       Description                                       Default   Strategy
%
%     strategy    Strategy designation                              stdCIS    All
%     inputGain   Input gain for scaling average impulse Leq of          1    All
%                 input signals (as calculated with splmeter.m) to LeqRef dB
%     LeqRef      Reference impulse Leq SPL in dB                       -9    All
%     eqcutoff    Equalization HP cutoff frequency in Hz              1200    All
%     eqord       Equalization HP filter order                           1    All
%     fl/BFmin    Lowest cutoff freq fl in Hz of BP filter bank, or    350    All
%                 lowest best frequency for DRNL filter bank (alternatively,
%                 fl/fu can be specified for DRNL filter bank also)
%     fu/BFmax    Highest cutoff freq fu in Hz of BP filter bank, or  5500    All
%                 highest best frequency for DRNL filter bank
%     gainBPin    Input gain into DRNL filter bank                       1    cpCIS, cpihcCIS, cpihclpCIS
%     bpord       BP filter order (for acoust. sim. in 'cp..' strategies) 6    All
%     gainBPout   BP filter output gain                                  1    All
%
%     DRNLParSet  'manual'    Parameter set values must be entered manually    cpCIS, cpihcCIS, cpihclpCIS
%                             through the command line. Default values are the DRNL
%                             regression parameters entered when option 'manual' was last
%                             used. Same as 'avghumregH' (see below), these parameters are
```

```
%                           for a "normalized" DRNL filter implementation, assuming a
%                           fractional number representation.
%                           This is the default used when DRNLParSet is ommitted.
%                'avghumregH' Parameter set developed during Enrique's visit in Nov 2002
%                           with no input scaling into DRNL filter, but an output scaling
%                           by H(BF). This "normalized" parameter set is based on
%                           'avghumreg' and moves the DRNL IO-curve into a range ]0, 1]
%                           suitable for an implementation using fractional numbers.
%                           This option must be used together with DRNLgain = 'normqb'
%                           or 'normqa'. See labbook 2, p. 24-27 for details.
%                 'avghumreg' Uses human cochlea parameters derived for average of
%                           6 normal hearing subjects using linear regression curve fitting,
%                           as described in Lopez-Poveda/Meddis (2001), "A human nonlinear
%                           cochlear filterbank". Note that unlike all other parameter sets,
%                           this one implies a cascade of 3 (as opposed to 2) gammatone
%                           filters in the linear DRNL path.
%                 'yohumreg'  Uses human cochlea parameters derived for subject YO
%                           using linear regression curve fitting, as described in
%                           Lopez-Poveda/Meddis (2001), "A human nonlinear cochlear filterbank".
%                 'avghum'    Uses human cochlea parameters derived for average of 6
%                           normal hearing subjects, as described in Lopez-Poveda/Meddis
%                           (2001), "A human nonlinear cochlear filterbank".
%                 'yohum'     Uses human cochlea parameters derived for subject YO,
%                           as described in Lopez-Poveda/Meddis (2001), "A human nonlinear
%                           cochlear filterbank".
%                 'human'     Uses parameters for human cochlea described in
%                           Lopez-Poveda/Meddis, "A computational model for simulating
%                           basilar-membrane nonlinearity in subjects with normal and
%                           impaired hearing".
%                 'animal'    Uses parameters for animal cochleae described in Meddis/O'Mard,
%                           2001, "A computational algorithm for computing nonlinear auditory
%                           frequency selectivity".
%                 NOTE:       With the last 4 options, for intermediate best frequencies BF not
%                           contained in the parameter sets, the DRNL filter parameters are
%                           derived through linear interpolation. Values outside the range of
%                           BF are extrapolated.
%     DRNLgain    'normqb'    DRNL output scaling of IO functions at BF so     cpCIS, cpihcCIS, cpihclpCIS
%                           that DRNL output is 1 for an input amplitude of 1
%                           across channels (output of 1 corresponds to MCL output).
%                           Requires  additional output scaling with H and LPcompgain
%                           and can only be used with DRNLParSet 'manual' or 'avghumregH'.
%                           For details see labbook 2, p. 24-27.
%                           This is the default used when DRNLgain is ommitted.
%                 'normqa'    DRNL output scaling of IO functions at BF so that THR is the same
%                           across channels and occurs at an input amp of ~1.58e-5. Requires
%                           additional output scaling with H and LPcompgain and can only be used
%                           with DRNLParSet 'manual' or 'avghumregH'.
%                 'norm'      In this implementation, the input gain g in the linear path is
%                           absorbed into the nonlinear compression coefficients a and b, by
%                           normalizing them to g = 1: a -> a/g, b -> b/g, g = 1. Thus, the
%                           linear gain g is 1 and no gain multiplication is necessary. Note
%                           that all signals through the stages of this DRNL filter are 1/g times
%                           the magnitude of the corresponding signals in function DRNLfilter.m,
%                           which includes the gain multiplication with g as found in
%                           Lopez-Poveda/Meddis.
%                           Thus, even if a is still > 1, due to the nonlinear compression
%                           taking min(a*x, b*x^v) for each sample and b being < 1, all
%                           signals should be < 1.
%                 'std'       Includes gain multiplication stage in the linear DRNL path, yielding
%                           a DRNL filter as described in Meddis/O'Mard and Lopez-Poveda/Meddis.
%   [ DRNLbwspec  'std'       With this option, DRNL bandwidth parameters      cpCIS, cpihcCIS, cpihclpCIS
%                           BWlin/BWnl are not interpreted as 3dB bandwidths
%                           of the gammatone filters of 1st order found in
%                           the DRNL filter cascades, but as "bandwidth" b, according
%                           to the impulse-response definition of a gammatone filter:
%                              h(t) := a*t^(ord-1) * exp(-2*pi*b*t) * cos(2*pi*fc*t + phi)
%                                      fc  ... center frequency [Hz]
%                                      b   ... bandwidth [Hz]
%                                      ord ... order
%                                      a   ... linear gain (usually a=1)
%                                      phi ... phase (usually phi=0)
%                           For that, function gammatone.m is called.
%                           This is the default for DRNLbwspec.
%                 '3dB'       With this option, DRNL bandwidth parameters BWlin/BWnl are
%                           interpreted as 3dB bandwidths of the gammatone filters of
%                           1st order in the DRNL cascades (gammatone3dB.m is called -
%                           see also gtBW3dB.m for more details).
%     DRNLiirimpl 'imp'       Uses the impulse invariant transformation        cpCIS, cpihcCIS, cpihclpCIS
%                           from continuous-time s-domain to discrete-time z-domain for
%                           implementing the gammatone filter as digital IIR filter.
%                           This is the default for DRNLiirimpl.
%                 'bil'       Uses the bilinear transformation from continuous-time s-domain
%                           to discrete-time z-domain for implementing the gammatone filter
%                           as digital IIR filter.
%                 'inv'       Uses the MATLAB generalized IIR filter design function invfeqz.m
%                           for implementing the gammatone filter as discrete-time IIR filter.
%                           NOTE: This function may yield non-monotonous frequency responses.  ]
%
%     rect        Envelope extraction rectifier ('full' or 'half')    'full'   stdCIS, cpCIS
%     lpcutoff    Envelope extraction LP cutoff frequency in Hz         200    stdCIS, stdihclpCIS, cpCIS, cpihclpCIS
%     lpord       Envelope extraction LP filter order                     4    stdCIS, stdihclpCIS, cpCIS, cpihclpCIS
%
%     IHCmodel    IHC model 'A' or 'B', as described in Meddis, 1986.  'B'    stdihcCIS, stdihclpCIS, cpihcCIS, cpihclpCIS
%     IHCParSet   'manual'      Parameter values must be entered manually      stdihcCIS, stdihclpCIS, cpihcCIS, cpihclpCIS
%                               through the command line. Default values are the
```

```
%                               parameters entered when option 'manual' was last used.
%                               This is the default when IHCParSet is ommitted.
%                 'Meddis90MSR' Model parameters for medium spontaneous rate
%                               auditory nerve fiber, published in JASA 87(4):
%                               pp. 1813-1816, 1990 (only for Model B).
%                 'Meddis90HSR' Model parameters for high spontaneous rate
%                               auditory nerve fiber, published in JASA 87(4):
%                               pp. 1813-1816, 1990 (only for Model B).
%                 'Meddis87'    Model parameters described in Meddis, 1987
%                               (only for Model B).
%                 'Meddis86'    Model parameters described in Meddis, 1986
%                               (for Model A or B).
%   [ IHCinput    'norm'        Normalized input signal (in range [-1,1]).
%                               IHCinput is predefined to 'norm' here.
%                 'model'       Input rectification/compression function k(s(t))
%                               is identical to the one in Meddis 1986, 1987,
%                               i.e. the input signal is unnormalized.          ]
%     gainLPin    Envelope extraction LP filter input gain             200    stdihclpCIS, cpihclpCIS
%
%     AdaptPar    Structure containing the following adaptation parameters:    stdhpCIS
%                 tau    Array of time constants in ms of HP filters  [1 10]
%                 gain   Gains applied to outputs of HP filters        [5 1]
%                 clamp  If 1, negative output of HP filter is clipped [1 0]
%     gainMAPin   Maplaw input gain                          1, 200 *ihcCIS    All
%     pwr         Maplaw power        -0.0001 stdCIS, 0.7 stdihc*, 1 cp*CIS    All
%   [ thr         THR value. Default value is 0 for having                0    All
%                 normalized output signals in range [THR,MCL] = [0,1].
%     mcl         MCL value. Default value is 1.                          1    All ]
%     Nmaplaw     Number of maplaw function samples                    1024    All
%     intp        Maplaw interpolation ('on' or 'off')                  'on'   All
%
%   specfile  Same as above
%
%   wavfile   Same as above
%
% Output argument:
%
%   stratpar  Cell array with 1 (unilateral AMP files) or 2 (bilateral AMp files) structures
%             containing all parameters used in signal processing strategy.
%             If appropriate, some or all of the following sub-structures are added
%             to structure elements as defined above:
%             DRNLRegrPar      Structure with parameter set vectors used in regression/interpolation
%                              to get actual parameter values at the given BFs.
%             DRNLParValues    Structure with actual parameters of all DRNL filters in bank.
%                              Structure elements are vectors containing DRNL parameters for
%                              channels 1:parspec.nchn.
%             IHCParValues     Structure containing all IHC parameter values.
%
% Reinhold Schatzer, 2002, 2003 RTI

% History:
%
% Version 2.4b, 06/25/03
%    - Updated function help to reflect possiblity to specify DRNL filter bank
%      frequency range as [BFmin, BFmax].
%
% Version 2.4, 06/19/03
%    - Added support for completely different strategies on left and right side
%      with bilateral spec files.
%    - Added strategy string field to strategy parameter structure (by maintaining
%      backward compatibility with older parameter structures).
%
% Version 2.3, 06/17/03
%    - Added strategy 'stdhpCIS'.
%    - Displaying progress bar before calculation of avg SPL and inputGain now.
%    - inputGain and LeqRef are saved to mat-file inputGain.mat in each wave file
%      directory now and can be retrieved later.
%
% Version 2.2, 05/15/03
%    - Optimized processing of AMP files for stereo noise left/right wave file
%      pairs, where left/right channel signal in noise left wave file is identical
%      to right/left channel signal in noise right wave file. If also the spec
%      file AND strategy parameters are identical on both sides, the right/left
%      side AMP sequence in the noise right AMP file will be identical to the
%      left/right side AMP sequence in the noise left AMP file. That is, the
%      noise right AMP file can be generated by simply switching left and right
%      AMP sequencies as calculated for the noise left AMP file.
%
% Version 2.1, 04/24/03
%    - Added possibility to scale wave file input signals to have average impulse Leq
%      (as calculated with splmeter.m) of LeqRef dB. LeqRef is defined as constant
%      in getStratParameters.m.
%    - Fixed "Index exceeds matrix dimension" error in writing stratpar to log file
%      for special case that mono or false stereo wave file is being processed as
%      1st input with identical stratpar on left and right side.
%
% Version 2.0, 04/04/03
%    - A raised cosine ramp of tramp msecs is now applied to wave file input signals
%      to get a smooth signal on/offset.
%    - Added support for bilateral MedEl and CI24M interfaces.
%    - For strategies 'cp*CIS', all DRNL parameters are written to log file now,
%      for strategies '*ihc*CIS', all IHC parameters.
%    - Input dialog for output directory is always openend now, using same default as before.
%    - Added progress bar.
%    - Added max rms channel output for each wave file processed to log file.
```

```
%    - Cell array stratpar with strategy parameter structures is saved to mat file now.
%
% Version 1.2, 10/22/02
%    - Added strategy functions stdihclpCIS and cpihclpCIS.
%    - Fixed DOS md call to work with spaces in directory path, too.
%
% Version 1.1, 10/03/02
%    - Updated function help text.
%    - Command window output recorded in file clipping.txt, if maplaw
%      clipping occurs during processing of input wave file.
%    - Added length of amp words in hex to logfile.txt.
%    - Sorting list of wav files now before processing them.
%
% Version 1.0, 09/25/02
%    - The beginning

%
% Constant definitions (change values here)
%
versionstr = '2.4';     % version string for amplitude sequence file header
DSPMemBufSize = hex2dec('80000');   % ADM56301 DRAM memory size in 24-bit words
% Number of bits for amplitude values in amplitude sequence output file for NewCS.
% Note that this number must be an integer fraction of 24, e.g. 8, 12 or 24.
nAmpSeqBitsNewCS = 12;
% Number of bits for amplitude values in amplitude sequence output file for MedEl/CI24M.
% The normalized amplitude is linear in the range [0,1]. For MedEl, 7-bit amplitude ranges
% [THR, MCL] are linear too, so that a linear transformation (MCL-THR)*normAmp + THR is fine
% and would require only 7 bits for the normalized amplitude.
% For the CI24M, only the nominal output current in uA is linear, but NOT the output amplitude
% specified in CL. Actually, output in uA increases exponentially with output in CL. Therefore,
% the normalized, linear amplitude in range [0,1] can only be transformed to a linear range
% [THR, MCL] with THR, MCL in uA, using a linear transformation (MCL-THR)*normAmp + THR. The
% linear transformation from normalized amplitude to [THR, MCL] cannot be done to THR, MCL in CL.
% Since the CI24M amplitude range in uA is [10, 1750] uA, 11 bits (2^12-1 = 2047) are required
% for the normalized amplitude to keep precision. Therefore, a linear range [0,1] using 12 bits
% for the normalized amplitudes is used with for MedEl and CI24M implants, to be able to interchange
% AMP files btw. these 2 implants, and also the NewCS (which uses 12-bit normalized amps, too).
nAmpSeqBitsBilat = 12;

% Parameters for raised cosine ramp applied to wave file signals for smooth signal onset/offset
tramp    = 10;          % duration of onset/offset ramp in msecs

%
% Check input arguments
%
if nargin >= 2 & iscell(varargin{2})
   syntax2 = 1;         % function called following convention 2 (stratpar{} provided)
   maxnargin = 5;
else
   syntax2 = 0;
   maxnargin = 4;
end
if nargin > maxnargin
   error('Too may input arguments.');
end
if nargin < maxnargin   % side argument
   side = 0;            % ask for side later, if stereo wave file is encountered
elseif ~ischar(varargin{maxnargin})
   error('Incorrect function call. Type ''help MakeAMP'' for help.');
else
   if strcmpi(side, 'right')
      sideidx = 2;      % right side signal is in column 2 of matrix returned by wavread()
   else
      sideidx = 1;      % left side signal is in column 1 of matrix returned by wavread()
   end
end
if nargin < maxnargin-1 % wavfile argument
   [wavfiles, wavfilepath] = uigetfiles('*.wav', 'Open wave file(s) to process (Ctrl/Shift click for multiple files)...');
   if isempty(wavfiles)
      error('Sorry, need wave file in order to continue.');
   end
   wavfiles = sort(wavfiles);
elseif ~ischar(varargin{maxnargin-1})
   error('Incorrect function call. Type ''help MakeAMP'' for help.');
else
   wavefile = varargin{maxnargin-1};
   if ~exist(wavefile, 'file')
      error(sprintf('Input wave file ''%s'' not found.', wavefile));
   end
   [path, name, ext] = fileparts(wavefile);  % separate path from wave file name
   wavfiles{1} = [name ext];     % save wave file name in cell array with 1 element (like uigetfiles does)
   wavfilepath = [path filesep]; % add directory separtator character '\' to end of path (like uigetfiles does)
   clear wavefile path name ext
end
if nargin < maxnargin-2    % specfile argument
   [specfilename, specfilepath] = uigetfile({'*.*','All Files (*.*)'}, 'Open ADSMonitor spec file ...');
   if specfilename == 0
      error('Sorry, need spec file in order to continue.');
   else
      specfile = [specfilepath specfilename];
      clear specfilepath specfilename
   end
elseif ~ischar(varargin{maxnargin-2})
   error('Incorrect function call. Type ''help MakeAMP'' for help.');
else
```

```
   specfile = varargin{maxnargin-2};
   if ~exist(specfile, 'file')
      error(sprintf('Input wave file ''%s'' not found.', specfile));
   end
end
if syntax2                % stratpar argument with cell array of parameter structure(s)
   stratpar = varargin{maxnargin-3};  % check later
end
if nargin > 0
   if ~iscell(varargin{1}) & ~ischar(varargin{1})  % strategy argument with cell array of strategy string(s)
      error('Incorrect function call. Type ''help MakeAMP'' for help.');
   else
      if iscell(varargin{1})
         strategy = varargin{1};
      else
         strategy{1} = varargin{1};
      end
      for i = 1:length(strategy)
         switch lower(strategy{i})
            case {'stdcis','cpcis','stdihccis','stdihclpcis','stdhpcis','cpihccis','cpihclpcis'}
            otherwise
               error('Incorrect function call. Type ''help MakeAMP'' for help.');
         end
      end
   end
end
DRNLParLogged = 0;   % flag to write detailed DRNL parameters to log file on their first availability
IHCParLogged  = 0;   % flag to write detailed IHC parameters to log file on their first availability

% Open diary file to record clipping warning messages. If no clipping
% occurs, diary file will be deleted, otherwise it will be moved to output
% directory later.
clippingFile = [getenv('TEMP') filesep 'clipping.txt'];  % clipping.txt in PC's TEMP directory path
diary off                 % make sure diary mode is off (i.e. that diary file is not read-only)
if exist(clippingFile, 'file')
   delete(clippingFile);   % make sure there is no old diary file, to which info would be appended
end
diary(clippingFile);
lastwarn('');  % clear last warning msg string to recognize if we get a warning

%
% Get parameters nchn, stimulation order and rate from spec file. Also, get
% actually implemented rate implRate, pulse onset times cyctim and header
% texts for AMP files.
%
[spec, implRate, cyctim, unilatAmpHeader, bilatAmpHeader] = SpecRead(specfile);
% Set bilatSpecFile flag
switch lower(spec.system)
   case 'new_currents'
      bilatSpecFile = 0;
%       interfacestr = 'New Current Source interface';
   case {'medel', 'ci24m'}
      if strcmpi(spec.side, 'bilateral')
         bilatSpecFile = 1;
         % If one side is delayed, check for how much. We only accept delays up to
         % the duration of 1 stimulation cycle on each side.
         eps = 1e-7;               % min. relative difference for compares of delay doubles
         cycdurus = 1e6./implRate;  % duration of stimulation cycles in microsecs
         if spec.rdelay < 0 & abs(spec.rdelay) - cycdurus(1) > eps  % left side delayed
            error(sprintf(['Left side delay is too big. Only delays up to the duration of 1 stimulation cycle are accepted.\n'
...
                           'Please reduce delay in spec file ''%s.'''], specfile));
         end
         if spec.rdelay > 0 & spec.rdelay - cycdurus(2) > eps       % right side delayed
            error(sprintf(['Right side delay is too big. Only delays up to the duration of 1 stimulation cycle are accepted.\n'
...
                           'Please reduce delay in spec file ''%s.'''], specfile));
         end
      else
         bilatSpecFile = 0;
      end
%       interfacestr = 'bilateral MedEl/CI24M interfaces';
end

% For bilateral processors, use same strategy on right side as on left side as
% a default, if only one strategy designation was provided as input argument.
if bilatSpecFile & nargin > 0 & length(strategy) == 1
   strategy{2} = strategy{1};
end

% If strategy parameter structure(s) are not provided as argument or are emtpy,
% get parameters from command line by calling getStratParameters() without
% 2nd argument. Otherwise, call getStratParameters() with structure(s) as 2nd
% argument and let the function check them and fill the rest with default values.
if ~syntax2
   % Get unilateral or left stratpar, respectively. Append  'ampl'/'ampr' to strategy string
   % to prevent getStratParameters from asking for NCHN (we'll get NCHN from spec file)
   if bilatSpecFile
      disp('Please enter processing strategy parameters for LEFT side first:');
      disp(' ');
   end
   if nargin == 0    % true, if no strategy input argument provided
      par = input(['Strategy - 1 = stdCIS: Regular CIS\n' ...
                   '           2 = stdihcCIS: CIS with Meddis IHC\n' ...
                   '           3 = stdihclpCIS: CIS with Meddis IHC and LP\n' ...
```

```
                  '           4 = stdhpCIS: CIS with Blake''s adaptation stage\n' ...
                  '           5 = cpCIS: CIS with DRNL filter bank\n' ...
                  '           6 = cpihcCIS: CIS with DRNL bank and Meddis IHC\n' ...
                  '           7 = cpihclpCIS: CIS with DRNL bank and Meddis IHC/LP (1): ']);
      if isempty(par)
         strategy{1} = 'stdCIS';
         par = 1;
      else
         switch par
            case 1
               strategy{1} = 'stdCIS';
            case 2
               strategy{1} = 'stdihcCIS';
            case 3
               strategy{1} = 'stdihclpCIS';
            case 4
               strategy{1} = 'stdhpCIS';
            case 5
               strategy{1} = 'cpCIS';
            case 6
               strategy{1} = 'cpihcCIS';
            case 7
               strategy{1} = 'cpihclpCIS';
            otherwise
               strategy{1} = 'stdCIS';
               par = 1;
         end
      end
   end
   stratpar{1} = getStratParameters(['ampl' strategy{1}]);  % left nchn is set later
   % Get right stratpar in case of bilateral spec file
   if bilatSpecFile
      disp(' ');
      disp('Please enter processing strategy parameters for RIGHT side:');
      disp(' ');
      if nargin == 0    % true, if no strategy input argument provided
         ans = input(sprintf(['Strategy - 1 = stdCIS: Regular CIS\n' ...
                        '           2 = stdihcCIS: CIS with Meddis IHC\n' ...
                        '           3 = stdihclpCIS: CIS with Meddis IHC and LP\n' ...
                        '           4 = stdhpCIS: CIS with Blake''s adaptation stage\n' ...
                        '           5 = cpCIS: CIS with DRNL filter bank\n' ...
                        '           6 = cpihcCIS: CIS with DRNL bank and Meddis IHC\n' ...
                        '           7 = cpihclpCIS: CIS with DRNL bank and Meddis IHC/LP (%i): '], par));
         if isempty(ans) | ans < 1 | ans > 7
            ans = par;  % default same as on left side
         end
         switch ans
            case 1
               strategy{2} = 'stdCIS';
            case 2
               strategy{2} = 'stdihcCIS';
            case 3
               strategy{2} = 'stdihclpCIS';
            case 4
               strategy{2} = 'stdhpCIS';
            case 5
               strategy{2} = 'cpCIS';
            case 6
               strategy{2} = 'cpihcCIS';
            case 7
               strategy{2} = 'cpihclpCIS';
         end
      end
      if strcmpi(strategy{1}, strategy{2})
         ans = input(sprintf(['Shall parameters just entered for LEFT side also be used on RIGHT side (y/n, default: y)? ']),
's');
         if isempty(ans)
            ans = 'y';
         end
         if strcmpi(ans(1), 'n')
            stratpar{2} = getStratParameters(['ampr' strategy{2}]);
         else
            stratpar{2} = stratpar{1}; % right nchn is set later
         end
      else
         stratpar{2} = getStratParameters(['ampr' strategy{2}]);
      end
   end   % if bilatSpecFile
else
   stratpar = varargin{2};
   if isstruct(stratpar{1})
      if ~isfield(stratpar{1}, 'strategy') % for compatibility with older parameter structures that do not contain strategy field
         stratpar{1}.strategy = strategy{1};
         stratpar{1} = orderfields(stratpar{1}, [length(fieldnames(stratpar{1})) 1:length(fieldnames(stratpar{1}))-1]);
      elseif ~strcmpi(strategy{1}, stratpar{1}.strategy)
         error(sprintf('Parameter structure in input argument ''stratpar{1}'' does not match selected strategy ''%s''.', ...
               strategy{1}));
      end
      stratpar{1} = getStratParameters(['ampl' strategy{1}], stratpar{1});  % check unilateral/left stratpar structure
   else
      error('Input argument stratpar{1} is not a structure. Type ''help MakeAMP'' for help');
   end
   if bilatSpecFile
      if isstruct(stratpar{2})
```

```
        if ~isfield(stratpar{2}, 'strategy') % for compatibility with older parameter structures that do not contain strategy
field
           stratpar{2}.strategy = strategy{2};
           stratpar{2} = orderfields(stratpar{2}, [length(fieldnames(stratpar{2})) 1:length(fieldnames(stratpar{2}))-1]);
        elseif ~strcmpi(strategy{2}, stratpar{2}.strategy)
           error(sprintf('Parameter structure in input argument ''stratpar{2}'' does not match selected strategy ''%s''.', ...
                 strategy{2}));
        end
        stratpar{2} = getStratParameters(['ampr' strategy{2}], stratpar{2}); % check right stratpar structure
     else
        error('Input argument stratpar{2} is not a structure. Type ''help MakeAMP'' for help');
     end
  end
end
stratpar{1}.nchn = spec.nchn(1);    % set nchn element in unilateral/left strategy structure
if bilatSpecFile
   stratpar{2}.nchn = spec.nchn(2); % set nchn element in right strategy structure
   % Make sure stratpar{2} has same inputGain settings as those entered so far for stratpar{1}
   stratpar{2}.inputGain = stratpar{1}.inputGain;
   stratpar{2}.LeqRef = stratpar{1}.LeqRef;
end

%
% Get name of and create output directory for amplitude sequence files.
% Default is a subdirectoy in the folder where wave files are taken from,
% named according to the following convention:
% <system>_<strategy><nchn>_<rate>pps                    for unilateral AMP files
% <system>_<strategy>_<Lnchn>L<Lrate>_<Rnchn>R<Rrate>_s|a  for synch|asynch bilateral AMP files
%     <system>       Implant interface system string ('NewCS', 'MedEl' or 'CI24M')
%     <strategy>     Strategy string ('stdCIS', 'cpCIS', 'stdihcCIS', 'stdihclpCIS', 'cpihcCIS' or 'cpihclpCIS')
%     <nchn>         Number of channels
%     <rate>         Nominal pulse rate in pps (can differ for simult./multirate processors)
% (    <pulsepattern> String describing stimulation pulse pattern ('intlvd' for regular,
%                     cond. or vcis processors, 'simult' for simult. pulse processor.
%                     Note that regular, conditioner and vcis processors with same nchn,
%                     stim. order and pulse rate can all use the same, identical
%                     amplitude sequence files. The ADSMonitor takes care of
%                     setting up the DSP channel data tables accordingly for the
%                     3 different types of processors).                              )
outputdir = wavfilepath;
switch lower(spec.system)
   case 'new_currents'
      outputdir = [outputdir 'NewCS'];
   otherwise
      outputdir = [outputdir spec.system];
end
if bilatSpecFile
   if strcmpi(spec.syncmode, 'no sync')
      cSync = 'a';
   else
      cSync = 's';
   end
   if strcmpi(strategy{1}, strategy{2})
      outputdir = sprintf('%s_%s_%uL%u_%uR%u_%c', outputdir, strategy{1}, stratpar{1}.nchn, spec.rate(1), ...
                          stratpar{2}.nchn, spec.rate(2), cSync);
   else
      outputdir = sprintf('%s_%s%uL%u_%s%uR%u_%c', outputdir, strategy{1}, stratpar{1}.nchn, spec.rate(1), ...
                          strategy{2}, stratpar{2}.nchn, spec.rate(2), cSync);
   end
else
   outputdir = sprintf('%s_%s%u_%upps', outputdir, strategy{1}, stratpar{1}.nchn, spec.rate);
end
%outputdir = sprintf('%s_%s%u_%upps_', outputdir, strategy{1}, stratpar{1}.nchn, spec.rate);
%if length(spec.order) == length([spec.order{:}])   % true for order vector with no simult. pulses
%   outputdir = [outputdir 'intlvd'];
%else
%   outputdir = [outputdir 'simult'];
%end
outputdir = inputdlg('Please enter full name of the output directory where AMP files will be saved:', ...
                     'Enter output directory name...', 1, {outputdir});
if isempty(outputdir)
   error('No output directory specified.');
end
outputdir = outputdir{1};
createdir = 1;
if exist(outputdir, 'dir')
   % Output directory already exists - check if it is empty, and if not, get new
   % output directory name
   if length(eval('dir(outputdir)')) > 2
      newoutdir = inputdlg(['This directory already exists and is not empty. Please enter a different ' ...
                            'output directory or path to prevent files from being overwritten:'], ...
                           'Enter different output directory...', 1, {outputdir});
      if isempty(newoutdir) | strcmpi(newoutdir{1}, outputdir)
         error('No new output directory specified.');
      else
         outputdir = newoutdir{1};
         clear newoutdir
      end
   else   % directory exists and is empty, so don't try to create it
      createdir = 0;
   end
end
if createdir
   [exitcode, message] = dos(sprintf('md "%s"', outputdir));
   if ~isempty(message)    % exitcode is 0 even in case of error
```

```
      error(sprintf('Error creating output directory %s:\n%s', outputdir, message));
   end
   clear exitcode message
end

%
% Calculate input gain for all wave files in selection to scale them to an
% average impulse Leq (as calculated with splmeter.m) of stratpar.LeqRef dB.
% This makes sure that wave file input signals that were mixed at different SNRs -
% which typically have different digital gains (or rms energies) - go into the
% CIS signal processing with approximately the same signal energy level. The average
% impulse Leq is used as signal level measure, because the impulse Leq is the most
% "constant" measure, in particular for sentences with pauses btw. words. Also,
% differences in Leq for wave files in quiet and in noise (at a given SNR, with
% the same output gain applied to signals in quiet and mixed with noise) are
% smallest for impulse Leq values.
%

% Show progress bar and disable TeX interpretation of progress bar text
hWaitBar = waitbar(0, '', 'Name', 'Generating AMP files...');
set(get(findobj(hWaitBar, 'Type', 'axes'), 'Title'), 'Interpreter', 'none');

if isempty(stratpar{1}.inputGain)   % inputGain is still empty if we need to adjust avg. Leq LeqRef dB
   % Look for inputGain*.mat file(s) in wave file directory and retrieve inputGain from there, if found
   inputGainFiles = dir([wavfilepath 'inputGain*.mat']);
   if length(inputGainFiles) > 0    % true if at least 1 file found
      if length(inputGainFiles) > 1
         aGainFiles = struct2cell(inputGainFiles);
         [sel, ok] = listdlg('Name', 'Select input gain file...', ...
                             'ListString', aGainFiles(1,:), ...
                             'SelectionMode', 'single', ...
                             'ListSize', [150 71]);
      else
         sel = 1;
      end
      % Get inputGain and corresponding LeqRef from file
      if ~isempty(sel)
         load([wavfilepath inputGainFiles(sel).name]);
         if isempty(inputGain) & ~isnumeric(inputGain) & isempty(LeqRef)
            error('Error retrieving inputGain/LeqRef from mat file in wave file directory.');
         end
         if LeqRef == stratpar{1}.LeqRef  % use loaded inputGain only if LeqRef corresponds
            stratpar{1}.inputGain = inputGain;
            if bilatSpecFile
               stratpar{2}.inputGain = inputGain;
            end
         end
      else  % user pressed cancel in list dialog
         warning('No input gain file selected. inputGain will be calculated from average SPL of wave file selection.');
      end
   end
   if isempty(stratpar{1}.inputGain)   % inputGain is still empty if LeqRef in mat-file did not match
      % Calculate slow, fast and impulse Leq SPLs for all wave files in selection.
      % Option 'mono' is used to include only mono and false stereo (left channel =
      % right channel) wave files the calculation of the Leq SPL values. That is,
      % for wave file with added directional noise, only wave file in quiet and with
      % noise from the front are included in determining the average Leq level.
      Leq = avgSPL('mono', 'off', wavfiles, wavfilepath);
      % Scaling factor to move average input impulse Leq to LeqRef dB. Is < 1, if Leq > LeqRef
      stratpar{1}.inputGain = 10^((stratpar{1}.LeqRef - Leq.imean)/20);
      if bilatSpecFile
         stratpar{2}.inputGain = stratpar{1}.inputGain;  % same Leq scaling parameters on both channels
         stratpar{2}.LeqRef    = stratpar{1}.LeqRef;     %  (Leq processed in 'mono' mode)
      end
      % Save inputGain and LeqRef to mat-file
      inputGain = stratpar{1}.inputGain;
      LeqRef = stratpar{1}.LeqRef;
      if exist([wavfilepath 'inputGain.mat'], 'file')
         directory = cd;
         cd(wavfilepath);
         [filename, pathname] = uiputfile('*.mat', 'Enter new file name for saving inputGain (and LeqRef)...');
         if isequal(filename,0) | isequal(pathname,0)
            warning('inputGain and LeqRef have not been saved to mat-file.')
         end
         cd(directory);
      else
         filename = 'inputGain.mat';
      end
      if ~isequal(filename,0)
         save([wavfilepath filename], 'inputGain', 'LeqRef');
      end
   end
   clear filename inputGain LeqRef
end   % if isempty(stratpar{1}.inputGain)

%
% Log everthing to string logstr, which is written to log file at end
%
clear logstr
% logstr = sprintf(['* Log file of amplitude sequence files generated for streaming mode\r\n' ...
%                   '* with %s on ADS56301 Lab System.\r\n'], interfacestr);
logstr = sprintf(['* Log file of amplitude sequence files generated for streaming mode\r\n' ...
                  '* on ADS56301 Lab System.\r\n']);
logstr = sprintf('%s* Files were generated by MakeAMP.m version %s.\r\n*\r\n', logstr, versionstr);
logstr = sprintf('%s* Input directory for wave files:\r\n*   %s\r\n*\r\n', logstr, wavfilepath);
```

```
if side  % true if side provided as function argument
   logstr = sprintf('%s* Signal channel processed in stereo wave files (selected via MakeAMP input argument):\r\n*
%s\r\n*\r\n',...
                    logstr, side);
end
logstr = sprintf('%s* Output directory for amp files:\r\n*   %s\r\n*\r\n', logstr, [outputdir filesep]);
logstr = sprintf('%s* ADSMonitor speech processor spec file scanned for processor parameters:\r\n*   %s\r\n*\r\n', ...
                 logstr, specfile);
logstr = sprintf('%s* Implant interface specification in spec file:\r\n*   %s\r\n*\r\n', ...
                 logstr, spec.system);
if bilatSpecFile
   idx = strfind(bilatAmpHeader, sprintf('*\r\n'));   % look for empty comment line
   logstr = sprintf('%s%s', logstr, bilatAmpHeader(1:idx+2));
else
   idx = strfind(unilatAmpHeader, sprintf('*\r\n'));   % look for empty comment line
   logstr = sprintf('%s%s', logstr, unilatAmpHeader(1:idx+2));
end
if bilatSpecFile
   if strcmpi(strategy{1}, strategy{2})
      logstr = sprintf('%s* Signal processing strategy:\r\n*   %s left and right side\r\n', logstr, strategy{1});
   else
      logstr = sprintf('%s* Signal processing strategy:\r\n*   %s left side, %s right side\r\n', logstr, strategy{1},
strategy{2});
   end
   if isequal(stratpar{1}, stratpar{2})   % true for identical stratpar on left and right
      logsides = 1;
      logsidestr = {' for left and right side'};
   else
      logsides = 2;
      logsidestr = {' for left side' ' for right side'};
   end
else
   logstr = sprintf('%s* Signal processing strategy:\r\n*   %s\r\n', logstr, strategy{1});
   logsides = 1;
   logsidestr = {''};
end
for i = 1:logsides
   logstr = sprintf('%s*\r\n* Strategy parameters%s:\r\n', logstr, logsidestr{i});
   fields = fieldnames(stratpar{i});   % cell array of structure field names
   values = struct2cell(stratpar{i});  % cell array of field values
   for j = 1:length(values)
      if isnumeric(values{j})
%         if length(values{j}) > 1   % vectors
%            vecstr1 = sprintf('*   %-13s: [', fields{j});
%            vecstr2 = sprintf(' %4g', values{j});
%            vecstr  = sprintf('%s%s ]\r\n', vecstr1, vecstr2);
%            logstr  = sprintf('%s%s', logstr, vecstr);
%            clear vecstr vecstr1 vecstr2
%          else                       % scalars
         logstr = sprintf('%s*   %-13s: %g\r\n', logstr, fields{j}, values{j});
%         end
      elseif isstruct(values{j})    % structures
         if strcmp(fields(j), 'AdaptPar') % do DRNL and/or IHC substructures later, when they contain valid data
            logstr = sprintf('%s*   %-13s:\r\n', logstr, fields{j}); % line '*   AdaptPar     :'
            subfields = fieldnames(values{j});        % cell array of substructure field names
            subvalues = struct2cell(values{j});       % cell array of field values
            for j = 1:length(subvalues)
%               if isnumeric(subvalues{j}) | islogical(subvalues{j})
%                  if length(subvalues{j}) > 1      % vectors
                  vecstr1 = sprintf('*          %-6s: [', subfields{j});
                  vecstr2 = sprintf('%3g ', subvalues{j});
                  logstr  = sprintf('%s%s%s]\r\n', logstr, vecstr1, vecstr2);
                  clear vecstr1 vecstr2
%                  else                           % scalars
%                     insstr = sprintf('%s*       %-6s: %g\r\n', insstr, subfields{j}, subvalues{j});
%                  end
%               else                              % strings
%                  insstr = sprintf('%s*         %-6s: ''%s''\r\n', insstr, subfields{j}, subvalues{j});
%               end
         end
      end
      else                        % strings
         logstr = sprintf('%s*   %-13s: ''%s''\r\n', logstr, fields{j}, values{j});
      end
   end
end
nMaxWavNameLength = size(strvcat(wavfiles),2);
logstr = sprintf('%s\r\nWave file%s-> Amp file (duration, max chnl output amp/rms, no. of pulses, no. of DSP words)\r\n', ...
                 logstr, 32 * ones(1, max(nMaxWavNameLength - 8, 1)));
logstr = sprintf('%s%s\r\n', logstr, 45 * ones(1, max(nMaxWavNameLength, 9) + 82));   % '-----' line

disp(' ');

%
% Prepare parameters for CIS sampling of channel output signals
%
switch lower(spec.system)
   case 'new_currents'
      % Set number of bits for amplitude sequence values
      nAmpSeqBits = nAmpSeqBitsNewCS;
      % Duration of stimulation cycle through channels is per definition 1/rate,
      % where rate is "pulse rate per channel":
      % Example of extended (multirate) stimulation cycle:  1  3  1  2  1  4  (1  3 ...)
      %                                                     <---   1/rate  --->
      cycdur = 1/implRate;                    % stimulation cycle duration in secs
```

```
      cyctim{1} = cyctim{1}*1e-6;              % pulse onset times must be in secs for sampling channel outputs
      orderlen = length(spec.order);           % number of non-simultaneous pulses in stimulation order cell array
      nAmpsPerCycle = length([spec.order{:}]);  % number of channel output amplitude values sampled in each stimulation cycle

   case {'medel', 'ci24m'}
      % Set number of bits for amplitude sequence values
      nAmpSeqBits = nAmpSeqBitsBilat;
      cycdur = 1./implRate;                    % duration of stimulation cycle(s) in secs
      cyctim{1} = cyctim{1}*1e-6;              % pulse onset times must be in secs for sampling channel outputs
      if bilatSpecFile
         cyctim{2} = cyctim{2}*1e-6;
         orderlen = [length(spec.order{1}) length(spec.order{2})]; % number of stimulation pulses within cycles
      else
         orderlen = length(spec.order{1});
      end
      nAmpsPerCycle = orderlen;  % number of channel output amplitude values sampled in each stimulation cycle
end
% Set fwrite precision string for given pulse amp bit length and number of amps per 24-bit DSP word
precstr = sprintf('ubit%u', nAmpSeqBits);
nAmpsPerDSPWord = 24/nAmpSeqBits;

%
% Loop over wave files, process them according to selected strategy, and save
% amplitude sequence files for given implant interface.
%

% Initialize boolean vector of files processed with all 0
bProcessedWavFiles = logical(zeros(1,length(wavfiles)));

for iwav = 1:length(wavfiles)

   %
   % Skip wave file, if AMP file was generated already, because of noise left/right
   % wave file pair and identical spec file and strategy parameters on both sides.
   %
   if bProcessedWavFiles(iwav)
      continue
   end

   %
   % Open wave file and check if stereo
   %
   [path, wavname, ext] = fileparts(wavfiles{iwav});
   if ~strcmpi(ext, '.wav')   % check if we have wav file
      warning(sprintf('File ''%s'' is not a wave file and is skipped.', wavfiles{iwav}));
      continue
   end
   [signal, fs, bits] = wavread([wavfilepath wavfiles{iwav}]);

   % Check if wave file signal is mono or stereo, and in the latter case also if
   % signals on left and right are identical. Also check for possibility to make
   % pair of noise left/right AMP files in one go.
   makeLRAMPpair = 0;      % clear flag to make pair of noise left/right AMP files in one go
   if size(signal,2) > 1 & ~all(signal(:,1) == signal(:,2))
      if bilatSpecFile     % true for stereo wave file with bilateral spec file
         amphdrwavinput = 'Stereo';
         makebilatAMP = 1; % set flag to make bilateral AMP file
         sameChnlOutputs = 0; % clear flag so that channel outputs are processed for both sides
         % Check for the special case of identical strategy parameters AND identical spec file
         % parameters (implRate, cyctim, order, and rdelay = 0 <=> unilatAmpHeader is not empty)
         % on left and right side, and a wave file selection containing noise left/right wave
         % file pairs, where typically channel signals are identical, but just switched. In
         % this case the noise right AMP file can be simply generated by switching left and
         % right AMP sequencies from the noise left AMP file.
         if isequal(stratpar{1}, stratpar{2}) & ~isempty(unilatAmpHeader) & ...
            lower(wavname(length(wavname))) == 'l' % wave files sorted -> L comes 1st
            % Open corresponding noise right wave file, if it is in selection
            wavnameR = [wavname(1:length(wavname)-1) 'R'];
            wavfileR = [wavnameR ext];
            wavfileRbidx = strcmpi(wavfiles, wavfileR);   % returns logical index vector
            if any(wavfileRbidx)
               signalR = wavread([wavfilepath wavfileR]);
               if size(signalR,2) > 1 & isequal(signal(:,1), signalR(:,2)) & isequal(signal(:,2), signalR(:,1))
                  makeLRAMPpair = 1;   % set flag to make pair of noise left/right AMP files in one go
               end
               clear signalR
            end
         end
      else                 % true for stereo wave file with unilateral spec file
         if ~side          % true if side was not provided as input argument
            sidestr = questdlg('Wave file selection contains stereo input file(s). Which channel shall be processed?', ...
                               'Channel selection', 'Left', 'Right', 'Left');
            if strcmp(sidestr,'Right')
               sideidx = 2;
            else
               sideidx = 1;
            end
            side = 1;      % process selected side with all stereo input files
            disp(sprintf('Selected %s side as channel to be processed with all stereo input wave files.', lower(sidestr)));
            logstr = sprintf('%s> Selected %s side as channel to be processed with all stereo input wave files.\r\n', ...
                             logstr, lower(sidestr));
         end
         amphdrwavinput = sprintf('Stereo -> %s channel processed', sidestr);
         signal = signal(:,sideidx); % process only data for selected side, i.e. mono input signal
         makebilatAMP = 0; % clear flag, so that we make unilateral AMP file
```

```
   end
else                    % true for mono wave file or false stereo wave file (identical signals on left and right)
   if size(signal,2) > 1
      signal = signal(:,1);   % reduce stereo input to mono if signals on left and right side are identical
      disp(sprintf(['> Stereo wave file %s has identical signals on left and right channel\n' ...
                    '> and will be processed in mono mode.'], wavfiles{iwav}));
      logstr = sprintf(['%s> Stereo wave file %s has identical signals on left and right channel and ' ...
                        'will be processed in mono mode.\r\n'], logstr, wavfiles{iwav});
      amphdrwavinput = 'Stereo with identical signals on left and right channel -> Mono';
   else
      amphdrwavinput = 'Mono';
   end
   if bilatSpecFile     % true for mono (or false stereo) wave file with bilateral spec file
      % Check for the following special cases with mono (or false stereo) wave file input
      % in combination with a bilateral spec file:
      % 1. Identical strategy and strategy parameters (including nchn) on both sides:
      %    In this case, channel output signals on left and right side will be identical.
      %    Thus, strategy function to calculate channel outputs needs to be called only once.
      % 2. Identical strategy, strategy parameters AND identical spec file parameters
      %    (implRate, cyctim, order, and rdelay = 0 <=> unilatAmpHeader is not empty):
      %    In this case, channel output signals on the 2 sides AND samples derived through
      %    CIS sampling of these channel outputs will be identical. Therefore, a unilateral
      %    AMP file with only one pulse amplitude sequence (for the 2 sides) will be generated.
      if isequal(stratpar{1}, stratpar{2})
         sameChnlOutputs = 1; % set flag to process channel outputs only once (case 1)
         if ~isempty(unilatAmpHeader)
            makebilatAMP = 0; % clear flag, so that we make unilateral AMP file (case 2)
         else
            makebilatAMP = 1; % set flag to make bilateral AMP file (case 1)
         end
      else
         makebilatAMP = 1;    % set flag to make bilateral AMP file
         sameChnlOutputs = 0; % clear flag so that channel outputs are processed for both sides
      end
   else                 % true for mono wave file with unilateral spec file
      makebilatAMP = 0; % clear flag, so that we make unilateral AMP file
   end
end

waitbar(sum(bProcessedWavFiles)/length(wavfiles), hWaitBar, sprintf('Processing wave file %s...', wavfiles{iwav}));
disp(sprintf('Processing wave file %s...', wavfiles{iwav}));

% Apply raised cosine ramp for smooth signal on/offset (multiply with scaling vector
% from 0 to 100% over tramp msecs at fs Hz)
if tramp > 0
   ramp = sin(linspace(0,pi/2,fs*tramp/1000)) .^ 2';
   rampgain = ones(size(signal,1),1);     % fill in scaling factors 1 first for length of signal
   rampgain(1:length(ramp)) = ramp;       % overwrite with startup ramp scaling factors
   rampgain(size(signal,1) - length(ramp) + 1 : size(signal,1)) = ...
      ramp(length(ramp):-1:1);            % overwrite with mirrored ramp noise scaling factors at end
   signal(:,1) = rampgain .* signal(:,1); % ramp left channel signal
   if size(signal,2) > 1
      signal(:,2) = rampgain .* signal(:,2); % ramp right channel signal for stereo input
   end
end

% Calculate length of stimulus and number of stimulus pulses
stimdur = size(signal,1)/fs;     % stimulus length in secs
nCycles = ceil(stimdur./cycdur); % number of stimulation cycles within stimulus length (rounded up)

% Check if binary amplitude sequence fits in DSP DRAM memory buffer
if makebilatAMP
   % Number of 24-bit DSP words required for bilateral AMP sequence
   nDSPWords = sum(nAmpsPerCycle.*nCycles/nAmpsPerDSPWord);
else
   % Number of 24-bit DSP words required for unilateral AMP sequence
   nDSPWords = nAmpsPerCycle(1)*nCycles(1)/nAmpsPerDSPWord;
end
if nDSPWords > DSPMemBufSize
   ratiofit = DSPMemBufSize/nDSPWords;
   warning(sprintf(['Wave file %s is skipped, because it is too long for ADS DRAM buffer\n'...
                    '         and given pulse rate. Only %u%% of the signal (%u ms) fits into the DRAM memory.\n'...
                    '         Try reducing wave signal duration or pulse rate.'], ...
                   wavfiles{iwav}, floor(ratiofit*100), round(ratiofit*stimdur*1000)));
   % Write to log file
   logstr = sprintf('%s> %s skipped, because too long for DRAM memory (only %u%% or %u ms fit into DRAM)!!!\r\n', ...
                    logstr, wavfiles{iwav}, floor(ratiofit*100), round(ratiofit*stimdur*1000));
   continue;   % process next wave file in for loop
end

%
% Process input signal according to processing strategy selected.
% Strategy functions return channel output signals, sampled at the same rate
% as the wave file input signal. Also, substructures in stratpar
% are filled with DRNL and/or IHC parameters that were used for
% the signal processing, where applicable.
%

% Process mono input or left channel with stereo input, using left (or unilateral) stratpar{1}
switch lower(strategy{1})
   case 'stdcis'
      [chnloutputs{1}, fs, stratpar{1}] = stdCIS(signal(:,1), fs, stratpar{1}, 'off');
   case 'cpcis'
      [chnloutputs{1}, fs, stratpar{1}] = cpCIS(signal(:,1), fs, stratpar{1}, 'off');
   case 'stdihccis'
```

```
      [chnloutputs{1}, fs, stratpar{1}] = stdihcCIS(signal(:,1), fs, stratpar{1}, 'off');
   case 'stdihclpcis'
      [chnloutputs{1}, fs, stratpar{1}] = stdihclpCIS(signal(:,1), fs, stratpar{1}, 'off');
   case 'stdhpcis'
      [chnloutputs{1}, fs, stratpar{1}] = stdhpCIS(signal(:,1), fs, stratpar{1}, 'off');
   case 'cpihccis'
      [chnloutputs{1}, fs, stratpar{1}] = cpihcCIS(signal(:,1), fs, stratpar{1}, 'off');
   case 'cpihclpcis'
      [chnloutputs{1}, fs, stratpar{1}] = cpihclpCIS(signal(:,1), fs, stratpar{1}, 'off');
end
% Make sure right stratpar gets all left stratpar updates (BF, DRNL/IHCParValues), if we
% have bilateral spec file and identical specs were entered for stratpar{1} and stratpar{2}.
% The update is done here to make sure stratpar{2} is also updated in the special case that
% a unilateral AMP file is being generated (bilateral spec file with identical stratpar,
% but mono or false stereo wave file input).
if bilatSpecFile & sameChnlOutputs
   stratpar{2} = stratpar{1};
end
% Process right channel output signals for bilateral AMP file using right stratpar{2} (if necessary)
if makebilatAMP
   if sameChnlOutputs   % true if right channel outputs are identical to left ones
      chnloutputs{2} = chnloutputs{1};
   else
      if size(signal,2) > 1
         sigidx = 2;    % process right channel of stereo input signal for right side
      else
         sigidx = 1;    % process mono input signal for right side (which has other parameters than left side)
      end
      switch lower(strategy{2})
         case 'stdcis'
            [chnloutputs{2}, fs, stratpar{2}] = stdCIS(signal(:,sigidx), fs, stratpar{2}, 'off');
         case 'cpcis'
            [chnloutputs{2}, fs, stratpar{2}] = cpCIS(signal(:,sigidx), fs, stratpar{2}, 'off');
         case 'stdihccis'
            [chnloutputs{2}, fs, stratpar{2}] = stdihcCIS(signal(:,sigidx), fs, stratpar{2}, 'off');
         case 'stdihclpcis'
            [chnloutputs{2}, fs, stratpar{2}] = stdihclpCIS(signal(:,sigidx), fs, stratpar{2}, 'off');
         case 'stdhpcis'
            [chnloutputs{2}, fs, stratpar{2}] = stdhpCIS(signal(:,sigidx), fs, stratpar{2}, 'off');
         case 'cpihccis'
            [chnloutputs{2}, fs, stratpar{2}] = cpihcCIS(signal(:,sigidx), fs, stratpar{2}, 'off');
         case 'cpihclpcis'
            [chnloutputs{2}, fs, stratpar{2}] = cpihclpCIS(signal(:,sigidx), fs, stratpar{2}, 'off');
      end
   end
end
if makebilatAMP
   disp(sprintf('   Maximum left-side channel output amplitude and rms from wave file %s:  %.3f, %.3f', ...
                wavfiles{iwav}, max(max(chnloutputs{1})), max(rms(chnloutputs{1}))));
   disp(sprintf('   Maximum right-side channel output amplitude and rms from wave file %s: %.3f, %.3f', ...
                wavfiles{iwav}, max(max(chnloutputs{2})), max(rms(chnloutputs{2}))));
else
   disp(sprintf('   Maximum channel output amplitude and rms from wave file %s: %.3f, %.3f', ...
                wavfiles{iwav}, max(max(chnloutputs{1})), max(rms(chnloutputs{1}))));
end

% Reserve column vector(s) for pulse amplitude values and intialize with 0.
% Length of vector(s) is an integer multiple of number of pulse amps per cycle.
% This guarantees a smooth transition to BKG stimulation at the end of the
% stimulus (amps at the stimulus end not filled in with sampled values
% remain at 0 and end up as THR pulses).
pulseamps{1} = zeros(nAmpsPerCycle(1).*nCycles(1), 1);    % unilateral or left side
if makebilatAMP
   pulseamps{2} = zeros(nAmpsPerCycle(2).*nCycles(2), 1); % right side
end

%
% Resample channel output signals at channel's pulse rate to get
% sequence of biphasic pulse amplitudes for amplitude sequence file.
% Resampling is done by going pulse by pulse through the channel output
% waveforms, following the given stimulation order. If the pulse onset
% lies within the waveform sampling interval [nT, (n+1)T[, the waveform
% sample at time nT is taken. This simple "flooring" algorithm comes
% closest to what happens in a real-time speech processor implementation
% in the DSP (option 'floor').
% Another option would be 'resample', which uses MATLAB's resample function to
% to resample the channout outputs at the overall stim. rate (function
% accepts only integer rates, i.e. a Med-El 18181.81... rate is rounded
% to 12*1515 = 18180 pps) - IS NOT IMPLEMENTED YET!
%

% Sample channel output waveforms into pulse amplitude vector.
% If pulse onset time t lies within the waveform sampling interval [nT, (n+1)T[,
% the waveform sample at time nT is taken. This simple "flooring" algorithm comes
% closest to what happens in a real-time speechprocessor implementation in the DSP.

% Sample left or unilateral side first
if makebilatAMP & spec.rdelay < 0   % check if left side is delayed
   time = abs(spec.rdelay*1e-6);    % set start time > 0 secs on left side if left side is delayed
else
   time = 0;                        % no delay on left side, so set start time to 0.
end
for i = 0:nCycles(1)-1                          % loop over stimulation cycles
   for j = 1:orderlen(1)                        % loop over interleaved, non-simult. pulse onsets within 1 cycle
      t = floor(fs*(time + cyctim{1}(j))) + 1;  % "time" index into matrix of channel output waveforms
```

```
%         t = round(fs*(time + cyctim{1}(j))) + 1;  % would be more exact than floor(), but not what DSP does
         if t <= size(chnloutputs{1}, 1)           % check if index t is in range
            if strcmp(spec.system, 'new_currents') % NewCS can have simultaneous pulses
               amp = chnloutputs{1}(t, spec.order{j}); % get channel output waveform sample(s) at time t (spec.order(j) is
                                                       %  vector of simult. channel numbers for simult. processor)
            else
               amp = chnloutputs{1}(t, spec.order{1}(j)); % get left/unilateral channel output sample at time t
                                                          %  from channel j in stimulation cycle
            end
         else                                      % index t exceeds channel output matrix...
            break;   % outer loop is always in last iteration here (leave last values in amp sequence that complete cycle at 0)
         end
         % Save sampled amplitude(s) in left/unilateral pulseamps vector
         if strcmp(spec.system, 'new_currents')    % NewCS can have simultaneous pulses
            pulseamps{1}(i*nAmpsPerCycle + length([spec.order{1:j-1}]) + 1 : ...
                         i*nAmpsPerCycle + length([spec.order{1:j}])) = amp;
         else
            pulseamps{1}(i*nAmpsPerCycle(1) + j) = amp;
         end
      end
      time = time + cycdur(1);
   end

   % Sample right side now for bilateral AMP files (not supported for NewCS interface)
   if makebilatAMP
      if spec.rdelay > 0           % check if right side is delayed
         time = spec.rdelay*1e-6;  % set start time > 0 secs on right side if right side is delayed
      else
         time = 0;                 % no delay on right side, so set start time to 0.
      end
      for i = 0:nCycles(2)-1                       % loop over stimulation cycles
         for j = 1:orderlen(2)                     % loop over interleaved, non-simult. pulse onsets within 1 cycle
            t = floor(fs*(time + cyctim{2}(j))) + 1;  % "time" index into matrix of channel output waveforms
%             t = round(fs*(time + cyctim{2}(j))) + 1;  % would be more exact than floor(), but not what DSP does
            if t <= size(chnloutputs{2}, 1)        % check if index t is in range
               amp = chnloutputs{2}(t, spec.order{2}(j)); % get right channel output sample at time t
                                                          %  from channel j in stimulation cycle
            else                                   % index t exceeds channel output matrix...
               break;   % outer loop is always in last iteration here (leave last values in amp sequence that complete cycle at
0)
            end
            % Save sampled amplitude in right pulseamps vector
            pulseamps{2}(i*nAmpsPerCycle(2) + j) = amp;
         end
         time = time + cycdur(2);
      end
   end

   %
   % Write everything to amplitude sequence file
   %

   % Add more header information as well as >length, >(l|r)pulsecount and >start bin tags
   % to amplitude header string returned by SpecRead.m.
   % Pulsecount corresponds to length of pulseamps sequence(s), where stimulation cycles are
   % completed at the end (i.e. nCycles * nAmpsPerCycles pulse amplitudes).
   if ~makebilatAMP  % unilateral AMP file header
      ampheader = [ ...
%          sprintf('* Amplitude sequence file for streaming mode with\r\n* %s on ADS56301 Lab System.\r\n', interfacestr) ...
         sprintf('* Amplitude sequence file for streaming mode on ADS56301 Lab System.\r\n') ...
         sprintf('* File contains unilateral amplitude sequence and was generated\r\n* by MakeAMP.m version %s.\r\n', versionstr)
...
         sprintf('* Input wave file name:\r\n*   %s\r\n', [wavfilepath wavfiles{iwav}]) ...
         sprintf('* Input wave file mode:\r\n*   %s\r\n', amphdrwavinput) ...
         sprintf('* Signal processing strategy:\r\n*   %s\r\n', strategy{1}) ...
         sprintf('* ADSMonitor speech processor spec file scanned for processor parameters:\r\n*   %s\r\n', specfile) ...
         unilatAmpHeader ];
      % Find start and end position of 2 ampheader lines containing system info (including CR+LF)
      [start, finish] = regexp(ampheader, '>system\s+(\w|\s)+\r\n');
      ampheader = [ ...
         ampheader(1:finish) ...
         sprintf('* Stimulus duration in msecs\r\n>length\r\n%u\r\n', round(stimdur*1000)) ...
         sprintf('* Number of deterministic pulse amplitudes in stimulus\r\n>pulsecount\r\n%u\r\n', length(pulseamps{1})) ...
         ampheader(finish+1:length(ampheader)) ...
         sprintf('* Pulse amplitude sequence in binary format\r\n>start bin\r\n') ];
   else              % bilateral AMP file header
      ampheader = [ ...
%          sprintf('* Amplitude sequence file for streaming mode with\r\n* %s on ADS56301 Lab System.\r\n', interfacestr) ...
         sprintf('* Amplitude sequence file for streaming mode on ADS56301 Lab System.\r\n') ...
         sprintf('* File contains bilateral amplitude sequences and was generated\r\n* by MakeAMP.m version %s.\r\n', versionstr)
...
         sprintf('* Input wave file name:\r\n*   %s\r\n', [wavfilepath wavfiles{iwav}]) ...
         sprintf('* Input wave file mode:\r\n*   %s\r\n', amphdrwavinput) ];
      if strcmpi(strategy{1}, strategy{2})
         ampheader = [ ...
            ampheader ...
            sprintf('* Signal processing strategy:\r\n*   %s left and right side\r\n', strategy{1}) ...
            sprintf('* ADSMonitor speech processor spec file scanned for processor parameters:\r\n*   %s\r\n', specfile) ...
            bilatAmpHeader ];
      else
         ampheader = [ ...
            ampheader ...
            sprintf('* Signal processing strategy:\r\n*   %s left side, %s right side\r\n', strategy{1}, strategy{2}) ...
            sprintf('* ADSMonitor speech processor spec file scanned for processor parameters:\r\n*   %s\r\n', specfile) ...
            bilatAmpHeader ];
```

```
   end
   % Find start and end position of 2 ampheader lines containing system info (including CR+LF)
   [start, finish] = regexp(ampheader, '>system\s+(\w|\s)+\r\n');
   ampheader = [ ...
      ampheader(1:finish) ...
      sprintf('* Stimulus duration in msecs\r\n>length\r\n%u\r\n', round(stimdur*1000)) ...
      sprintf('* Number of deterministic pulse amplitudes on each side\r\n>lpulsecount\r\n%u\r\n', length(pulseamps{1})) ...
      sprintf('>rpulsecount\r\n%u\r\n', length(pulseamps{2})) ...
      ampheader(finish+1:length(ampheader)) ...
      sprintf('* Bilateral pulse amplitude sequences in binary format\r\n>start bin\r\n') ];
end

% Open amp output file in binary write mode using big-endian format (high byte first)
fid = fopen([outputdir filesep wavname '.amp'], 'wb', 'ieee-be');
if fid == -1
   error(sprintf('Couldn''t open output file %s.', [wavname '.amp']));
end

% Write header information to amplitude sequence file first
fwrite(fid, ampheader);

% Round amplitude sequence(s) to unsigned integers of nAmpSeqBits bits and write to file
% by packing values into nAmpSeqBits-bit unsigned int format

% Write left/unilateral pulseamps to file first
if any(pulseamps{1} < 0)
   warning(sprintf(['Found values < 0 in unilateral or left amplitude sequence derived from input wave file\n' ...
                    '%s. These amplitude values were clipped to 0!'], wavfiles{iwav}));
   pulseamps{1} = max(pulseamps{1}, 0);
end
if any(pulseamps{1} > 1)
   warning(sprintf(['Found values > 1 in unilateral or left amplitude sequence derived from input wave file\n' ...
                    '%s. These amplitude values were clipped to 1!'], wavfiles{iwav}));
   pulseamps{1} = min(pulseamps{1}, 1);
end
% Write sequence of integer amps to file. If the last byte is not full, it would
% be filled with trailing 0-bits only when file is closed (that's true also for
% very large file size - tested file size of 15*10^6 bytes).
fwrite(fid, round(pulseamps{1} * (2^nAmpSeqBits - 1)), precstr);
% Fill up last 24-bit word in amplitude sequence with 0-amps. If a 0-amps is (are) added,
% they are NOT reflected in >pulsecount, which is still an integer multiple of nAmpsPerCycle.
nfill = nAmpsPerDSPWord*ceil(length(pulseamps{1})/nAmpsPerDSPWord) - length(pulseamps{1});
for i = 1:nfill   % fill up 24-bit word with nfill zero pulse amps
   fwrite(fid, 0, precstr);
end

% Write right pulseamps to bilateral AMP file now
if makebilatAMP
   if any(pulseamps{2} < 0)
      warning(sprintf(['Found values < 0 in right amplitude sequence derived from input wave file\n' ...
                       '%s. These amplitude values were clipped to 0!'], wavfiles{iwav}));
      pulseamps{2} = max(pulseamps{2}, 0);
   end
   if any(pulseamps{2} > 1)
      warning(sprintf(['Found values > 1 in right amplitude sequence derived from input wave file\n' ...
                       '%s. These amplitude values were clipped to 1!'], wavfiles{iwav}));
      pulseamps{2} = min(pulseamps{2}, 1);
   end
   % Write sequence of integer amps to file. If the last byte is not full, it would
   % be filled with trailing 0-bits only when file is closed (that's true also for
   % very large file size - tested file size of 15*10^6 bytes).
   fwrite(fid, round(pulseamps{2} * (2^nAmpSeqBits - 1)), precstr);
   % Fill up last 24-bit word in amplitude sequence with 0-amps. If a 0-amps is (are) added,
   % they are NOT reflected in >pulsecount, which is still an integer multiple of nAmpsPerCycle.
   nfill = nAmpsPerDSPWord*ceil(length(pulseamps{2})/nAmpsPerDSPWord) - length(pulseamps{2});
   for i = 1:nfill   % fill up 24-bit word with nfill zero pulse amps
      fwrite(fid, 0, precstr);
   end
end

% Close amplitude sequence file and set flag that corresponding wave file was processed
fclose(fid);
bProcessedWavFiles(iwav) = 1;

% If we have noise left/right wave file pair with identical spec file and
% strategy parameters on left and right side, and if we just generated the
% noise left AMP file, make noise right AMP file now by simply switching
% left and right side AMP sequencies.
if makeLRAMPpair

   % Display info (logged later, after everything's logged for noise left file)
   waitbar(sum(bProcessedWavFiles)/length(wavfiles), hWaitBar, sprintf('Processing wave file %s...', wavfileR));
   disp(sprintf(['> Stereo wave file %s results in identical bilateral AMP sequence as wave file %s,\n' ...
                 '> except that left and right AMP sequences are switched. Thus, AMP file for %s is generated by\n' ...
                 '> simply switching left and right AMP sequences that were calculated for wave file %s.'], ...
                 wavfileR, wavfiles{iwav}, wavfileR, wavfiles{iwav}));
   disp(sprintf('Processing wave file %s...', wavfileR));
   disp(sprintf('   Maximum left-side channel output amplitude and rms from wave file %s:  %.3f, %.3f', ...
                wavfileR, max(max(chnloutputs{2})), max(rms(chnloutputs{2}))));
   disp(sprintf('   Maximum right-side channel output amplitude and rms from wave file %s: %.3f, %.3f', ...
                wavfileR, max(max(chnloutputs{1})), max(rms(chnloutputs{1}))));

   % Replace file name in ampheader
   ampheader = strrep(ampheader, wavfiles{iwav}, wavfileR);
```

```
      % Open amp output file in binary write mode using big-endian format (high byte first)
      fid = fopen([outputdir filesep wavnameR '.amp'], 'wb', 'ieee-be');
      if fid == -1
         error(sprintf('Couldn''t open output file %s.', [wavnameR '.amp']));
      end

      % Write header information to amplitude sequence file first
      fwrite(fid, ampheader);

      % Round amplitude sequence(s) to unsigned integers of nAmpSeqBits bits and write to file
      % by packing values into nAmpSeqBits-bit unsigned int format

      % Write right pulseamps of noise left file as left side AMP sequence to noise right
      % AMP file, i.e. write them to AMP file first
      fwrite(fid, round(pulseamps{2} * (2^nAmpSeqBits - 1)), precstr);
      % Fill up last 24-bit word in amplitude sequence with 0-amps. If a 0-amps is (are) added,
      % they are NOT reflected in >pulsecount, which is still an integer multiple of nAmpsPerCycle.
      nfill = nAmpsPerDSPWord*ceil(length(pulseamps{2})/nAmpsPerDSPWord) - length(pulseamps{2});
      for i = 1:nfill   % fill up 24-bit word with nfill zero pulse amps
         fwrite(fid, 0, precstr);
      end
      % Write left pulseamps of noise left file as right side AMP sequence to noise right AMP file
      fwrite(fid, round(pulseamps{1} * (2^nAmpSeqBits - 1)), precstr);
      % Fill up last 24-bit word in amplitude sequence with 0-amps. If a 0-amps is (are) added,
      % they are NOT reflected in >pulsecount, which is still an integer multiple of nAmpsPerCycle.
      nfill = nAmpsPerDSPWord*ceil(length(pulseamps{1})/nAmpsPerDSPWord) - length(pulseamps{1});
      for i = 1:nfill   % fill up 24-bit word with nfill zero pulse amps
         fwrite(fid, 0, precstr);
      end

      % Close amplitude sequence file and set flag that corresponding wave file was processed
      fclose(fid);
      bProcessedWavFiles(wavfileRbidx) = 1;

   end   % if makeLRAMPpair

   % Insert data in vector BF and substructures DRNLRegrPar and DRNLParValues to log file for
   % strategies cp*CIS, now that they were returned in stratpar by in the strategy function call.
   if ~DRNLParLogged & (strcmpi(strategy{1}(1:2), 'cp') | strcmpi(strategy{2}(1:2), 'cp'))
      DRNLParLogged = 1;
      cpidx = find(strncmpi(strategy,'cp',2) == 1);   % vector of indices in stratpar{} where cp*CIS parameter structure(s)
is(are)
      % Determine number of how many cp*CIS parameter structures were logged (1, if stratpar{1} = stratpar{2})
      ncplog = length(strfind(logstr, '*   DRNLParSet'));
      for i = 1:ncplog          % loop over how often BF info has to be inserted
         BFstr1 = sprintf('*   %-13s: [', 'BF');
         BFstr2 = sprintf(' %d', round(stratpar{cpidx(i)}.BF));
         BFstr  = sprintf('%s%s ]\r\n', BFstr1, BFstr2);
         offset = strfind(logstr, '*   DRNLParSet');  % find offset(s) of line start(s) where BF will be inserted
         logstr = sprintf('%s%s%s', logstr(1:offset(i)-1), BFstr, logstr(offset(i) : length(logstr)));
         clear BFstr BFstr1 BFstr2
      end
      for i = 1:ncplog  % loop over how often DRNLRegrPar info has to be inserted (only once, if stratpar{1} = stratpar{2})
         clear insstr   % make sure string that will be put together to be inserted in logstr is empty
         insstr = sprintf('*   %-13s:\r\n', 'DRNLRegrPar');
         subfields = fieldnames(stratpar{cpidx(i)}.DRNLRegrPar);  % cell array of substructure field names
         subvalues = struct2cell(stratpar{cpidx(i)}.DRNLRegrPar); % cell array of field values
         for j = 1:length(subvalues)
%            if isnumeric(subvalues{j})
%               if length(subvalues{j}) > 1      % vectors
            vecstr1 = sprintf('*          %-6s: [', subfields{j});
            vecstr2 = sprintf('%8.5g ', subvalues{j});
            insstr  = sprintf('%s%s%s]\r\n', insstr, vecstr1, vecstr2);
            clear vecstr1 vecstr2
%               else                             % scalars
%                  insstr = sprintf('%s*       %-6s: %g\r\n', insstr, subfields{j}, subvalues{j});
%               end
%            else                               % strings
%               insstr = sprintf('%s*         %-6s: ''%s''\r\n', insstr, subfields{j}, subvalues{j});
%            end
         end
         % Insert string insstr now
         offset = strfind(logstr, '*   bpord'); % find offset(s) of line start(s) where DRNLRegrPar will be inserted
         cpoffset = offset(cpidx(i));           % make sure '* bpord' offset is that where cp*CIS structure is logged,
                                                %  in case both left and right structures are logged and only one is for cp*CIS
         logstr  = sprintf('%s%s%s', logstr(1:cpoffset-1), insstr, logstr(cpoffset : length(logstr)));
      end
      for i = 1:ncplog  % loop over how often DRNLRegrPar info has to be inserted (only once, if stratpar{1} = stratpar{2})
         clear insstr   % make sure string that will be put together to be inserted in logstr is empty
         insstr = sprintf('*   %-13s:\r\n', 'DRNLParValues');
         subfields = fieldnames(stratpar{cpidx(i)}.DRNLParValues);   % char array of substructure field names (filled with
spaces)
         subvalues = struct2cell(stratpar{cpidx(i)}.DRNLParValues);  % cell array of field values
         for j = 1:length(subvalues)
            vecstr1 = sprintf('*      %-10s: [', subfields{j});
            vecstr2 = sprintf('%8.5g ', subvalues{j});
            insstr  = sprintf('%s%s%s]\r\n', insstr, vecstr1, vecstr2);
            clear vecstr1 vecstr2
         end
         % Insert string insstr now
         offset = strfind(logstr, '*   bpord'); % find offset(s) of line start(s) where DRNLParValues will be inserted
         cpoffset = offset(cpidx(i));           % make sure '* bpord' offset is that where cp*CIS structure is logged,
                                                %  in case both left and right structures are logged and only one is for cp*CIS
         logstr  = sprintf('%s%s%s', logstr(1:cpoffset-1), insstr, logstr(cpoffset : length(logstr)));
      end
```

```
   end   % if ~DRNLParLogged & (strcmpi(strategy{1}(1:2), 'cp') | strcmpi(strategy{2}(1:2), 'cp'))

   % Insert data in substructure IHCParValues to log file for strategies *ihc*CIS,
   % now that they were returned in stratpar by in the strategy function call.
   if ~IHCParLogged & (~isempty(strfind(lower(strategy{1}), 'ihc')) | ~isempty(strfind(lower(strategy{2}), 'ihc')))
      IHCParLogged = 1;
      % Make vector of indices in stratpar{} where *ihc*CIS parameter structure(s) is(are)
      if ~isempty(strfind(lower(strategy{1}), 'ihc'))
         ihcidx = 1;
      else
         ihcidx = [];
      end
      if ~isempty(strfind(lower(strategy{2}), 'ihc'))
         ihcidx = [ihcidx 2];
      end
      offset = strfind(logstr, '*   IHCinput'); % find offset(s) of line(s) IHCinput to insert IHCParValues in following line
      for i = 1:length(offset)   % loop over how often IHCParValues info has to be inserted (only once, if stratpar{1} =
stratpar{2})
         nextlines = strfind(logstr(offset(i):length(logstr)), '*   '); % find offset of lines after IHCinput
         offset(i) = offset(i) + nextlines(2) - 1; % offset of 1st line after line IHCinput
         clear insstr           % make sure string that will be put together to be inserted in logstr is empty
         insstr = sprintf('*   %-13s:\r\n', 'IHCParValues');
         subfields = fieldnames(stratpar{ihcidx(i)}.IHCParValues);  % char array of substructure field names (filled with spaces)
         subvalues = struct2cell(stratpar{ihcidx(i)}.IHCParValues); % cell array of field values
         for j = 1:length(subvalues)
            if ischar(subvalues{j})
               insstr  = sprintf('%s*      %-10s: ''%s''\r\n', insstr, subfields{j}, subvalues{j});
            else
               insstr  = sprintf('%s*      %-10s: %.14g\r\n', insstr, subfields{j}, subvalues{j});
            end
         end
         % Insert string insstr now
         logstr  = sprintf('%s%s%s', logstr(1:offset(i)-1), insstr, logstr(offset(i) : length(logstr)));
      end
   end   % if ~IHCParLogged & (strfind(lower(strategy{1}), 'ihc') | strfind(lower(strategy{2}), 'ihc'))

   % Add info about processed wave file and uni/bilateral amplitude sequence to log string
   if ~makebilatAMP
      logstr = sprintf('%s%s%s -> %s%s (%u ms, %.3f/%.3f, %u = 0x%X pulses, %u = 0x%X DSP words)\r\n', ...
                       logstr, wavfiles{iwav}, 32 * ones(1, max(nMaxWavNameLength, 9) - length(wavfiles{iwav})), ...
                       [wavname '.amp'], 32 * ones(1, max(nMaxWavNameLength, 8) - length(wavfiles{iwav})), ...
                       round(stimdur*1000), max(max(chnloutputs{1})), max(rms(chnloutputs{1})), length(pulseamps{1}), ...
                       length(pulseamps{1}), ceil(length(pulseamps{1})/2), ceil(length(pulseamps{1})/2));
   else
      % Left-side amplitude sequence stats
      logstr = sprintf('%s%s%s -> %s%s (%u ms, %.3f/%.3f, %u = 0x%X pulses, %u = 0x%X DSP words left)\r\n', ...
                       logstr, wavfiles{iwav}, 32 * ones(1, max(nMaxWavNameLength, 9) - length(wavfiles{iwav})), ...
                       [wavname '.amp'], 32 * ones(1, max(nMaxWavNameLength, 8) - length(wavfiles{iwav})), ...
                       round(stimdur*1000), max(max(chnloutputs{1})), max(rms(chnloutputs{1})), length(pulseamps{1}), ...
                       length(pulseamps{1}), ceil(length(pulseamps{1})/2), ceil(length(pulseamps{1})/2));
      % Right-side amplitude sequence stats
      logstr = sprintf('%s%s (%u ms, %.3f/%.3f, %u = 0x%X pulses, %u = 0x%X DSP words right)\r\n', ...
                       logstr, 32 * ones(1, max(nMaxWavNameLength, 9) + 4 + max(nMaxWavNameLength, 8)), ...
                       round(stimdur*1000), max(max(chnloutputs{2})), max(rms(chnloutputs{2})), length(pulseamps{2}), ...
                       length(pulseamps{2}), ceil(length(pulseamps{2})/2), ceil(length(pulseamps{2})/2));
   end
   % If we have noise left/right wave file pair with identical spec file and
   % strategy parameters on left and right side, and thus just generated the
   % AMP file pairs, log info for noise right AMP file now.
   if makeLRAMPpair
      logstr = sprintf(['%s> Stereo wave file %s results in identical bilateral AMP sequence as wave file %s,\r\n' ...
                        '> except that left and right AMP sequences are switched. Thus, AMP file for %s is generated by\r\n' ...
                        '> simply switching left and right AMP sequences that were calculated for wave file %s.\r\n'], ...
                        logstr, wavfileR, wavfiles{iwav}, wavfileR, wavfiles{iwav});
      % Left-side amplitude sequence stats
      logstr = sprintf('%s%s%s -> %s%s (%u ms, %.3f/%.3f, %u = 0x%X pulses, %u = 0x%X DSP words right)\r\n', ...
                       logstr, wavfileR, 32 * ones(1, max(nMaxWavNameLength, 9) - length(wavfileR)), ...
                       [wavnameR '.amp'], 32 * ones(1, max(nMaxWavNameLength, 8) - length(wavfileR)), ...
                       round(stimdur*1000), max(max(chnloutputs{2})), max(rms(chnloutputs{2})), length(pulseamps{2}), ...
                       length(pulseamps{2}), ceil(length(pulseamps{2})/2), ceil(length(pulseamps{2})/2));
      % Right-side amplitude sequence stats
      logstr = sprintf('%s%s (%u ms, %.3f/%.3f, %u = 0x%X pulses, %u = 0x%X DSP words left)\r\n', ...
                       logstr, 32 * ones(1, max(nMaxWavNameLength, 9) + 4 + max(nMaxWavNameLength, 8)), ...
                       round(stimdur*1000), max(max(chnloutputs{1})), max(rms(chnloutputs{1})), length(pulseamps{1}), ...
                       length(pulseamps{1}), ceil(length(pulseamps{1})/2), ceil(length(pulseamps{1})/2));
      % Clear flag to make LR AMP file pair
      makeLRAMPpair = 0;
   end

   clear chnloutputs pulseamps

   waitbar(sum(bProcessedWavFiles)/length(wavfiles)); % update progress bar

end   % for iwav = 1:length(wavfiles)

%
% Put out progress info and close log file
%
disp( '------------------------------------------------------------------------------' );
disp( sprintf( ' Processing completed on %s', datestr(now) ) );
disp( '------------------------------------------------------------------------------' );
logstr = sprintf(['%s\r\n-------------------------------------------------------------------------------------------------\r\n' ...
                  ' Processing completed on %s\r\n' ...
                  '-------------------------------------------------------------------------------------------------\r\n'], ...
                  logstr, datestr(now));
```

```
%
% Open log file and write out logstr
%
logid = fopen([outputdir filesep 'logfile.txt'], 'w+');
if logid == -1
   error(sprintf('Error opening log file %s.', [outputdir filesep 'logfile.txt']));
end
fwrite(fid, logstr);
fclose(logid);

%
% Close diary and move it to output directory
%
diary off
msgstr = lastwarn;
if ~isempty(msgstr)  % if true, we got warning during processing, so don't delete diary file
   movefile(clippingFile, [outputdir filesep]);
else
   delete(clippingFile);
end

%
% Save stratpar in output directory and close progress bar window
%
save([outputdir filesep 'stratpar'], 'stratpar');
close(hWaitBar);

% Avoid display of return value ans if MakeAMP is called without output argument and trailing ';'
if nargout < 1
   clear stratpar
end
```