CHAPRO

Compression Hearing-Aid Processing Library
API Documentation

Boys Town National Research Hospital
25 September 2019

TABLE OF CONTENTS

CHAPRO LIBRARY OVERVIEW

CHAPRO is a library of functions that may be used to implement simulations of compression hearing-aid signal processing. Four different types of signal processing strategies are included:

(1) FIR filter-bank frequency analysis with automatic gain control;

(2) IIR filter-bank frequency analysis with automatic gain control;

(3) Complex FIR filter-bank frequency analysis with instantaneous compression;

(4) Complex IIR filter-bank frequency analysis with instantaneous compression.

A modular design has been adopted to facilitate replacement of library functions with alternative signal-processing implementations. Each of the six major modules contains (1) a *preparation* function that allocates memory and initializes variables and (2) one or more *processing* functions that perform signal processing.

1. FIR filter-bank
    a. `firfb_prepare`
    b. `firfb_analyze`
    c. `firfb_synthesize`
2. IIR filter-bank
    a. `iirfb_design`
    b. `iirfb_prepare`
    c. `iirfb_analyze`
    d. `iirfb_synthesize`
3. Automatic gain control
    a. `agc_prepare`
    b. `agc_input`
    c. `agc_channel`
    d. `agc_output`
4. Adaptive feedback cancelation
    a. `afc_prepare`
    b. `afc_input`
    c. `afc_output`
5. Complex IIR filter-bank
    a. `ciirfb_design`
    b. `ciirfb_prepare`
    c. `ciirfb_analyze`
    d. `ciirfb_synthesize`
6. Instantaneous compression
    a. `icmp_prepare`
    b. `icmp_process`

For each module, variables are initialized and associated data memory is allocated by the preparation functions. This storage is combined into a single data structure to facilitate creation of firmware for real-time implementation on signal-processing hardware that might not have an operating system. The CHAPRO library includes a function that generates a C-code representation of this initialized data.

- `data_gen`

For desktop simulation, the CHAPRO library includes core functions for memory allocation and disposal.

- `prepare`
- `allocate`
- `cleanup`

Functions for FFT of real signals are included among the core functions.

- `fft_rc`
- `fft_cr`

Finally, the CHAPRO library includes a function that returns a version description string.

- `version`

To simulate complex IIR filter-bank frequency analysis with instantaneous compression, variable initialization and memory allocation is performed by calling the following functions.

- `ciirfb_prepare`
- `icmp_prepare`

Subsequent signal processing is performed by calling the following functions.

- `ciirfb_analyze`
- `icmp_process`
- `ciirfb_synthesize`

Several examples test basic aspects of these functions.

- `tst_cifa` – tests complex IIR filter-bank analysis
- `tst_cifio` – tests simple waveform complex IIR processing
- `tst_cifsc` – tests simple waveform complex IIR processing with soundcard

To simulate FIR filter-bank frequency analysis with AGC compression, variable initialization and memory allocation is performed by calling the following prepare functions.

- `firfb_prepare`
- `agc_prepare`

Subsequent signal processing is performed by calling the following process functions.

- `agc_input`
- `firfb_analyze`
- `agc_channel`
- `firfb_synthesize`

- `agc_output`

Several examples test basic aspects of these functions.

- `tst_ffa` – tests filter-bank analysis
- `tst_ffio` – tests simple waveform FIR processing
- `tst_ffsc` – tests speech-waveform FIR & AGC with soundcard

To simulate IIR filter-bank frequency analysis with AGC compression and adaptive feedback cancelation, variable initialization and memory allocation is performed by calling the following functions.

- `afc_prepare`
- `agc_prepare`
- `iirfb_prepare`

Subsequent signal processing is performed by calling the following functions.

- `afc_input`
- `agc_input`
- `iirfb_analyze`
- `agc_channel`
- `iirfb_synthesize`
- `agc_output`
- `afc_output`

Several examples test basic aspects of these functions.

- `tst_ifa` – tests filter-bank analysis
- `tst_ifio` – tests simple waveform IIR processing
- `tst_ifsc` – tests speech-waveform IIR & AGC with soundcard
- `tst_iffb` – tests speech-waveform IIR & AFC with soundcard
- `tst_gha` – tests speech-waveform IIR & AFC & AGC with soundcard

All examples require the SIGPRO library from BTNRH (http://audres.org/rc/sigpro). The soundcard examples also require the ARSC library (http://audres.org/rc/arsc).

CHAPRO FUNCTION DESCRIPTIONS

cha_allocate

Allocates memory attached to CHAPRO data structure.

(void *) **cha_allocate**(CHA_PTR **cp**, int **cnt**, int **siz**, int **idx**)

Function arguments

cp	pointer to CHAPRO data structure
cnt	Number of elements to allocate.
siz	Size of each element.
idx	Index into CHAPPRO data structure.

Return Value

Pointer to the allocated memory.

Remarks

A pointer to the allocated memory is stored in the CHAPRO data structure at the location specified by **idx**.

See Also

cha_cleanup

cha_cleanup
Frees all memory attached to CHAPRO data structure.

(void) **cha_cleanup**(CHA_PTR **cp**)

Function arguments

cp pointer to CHAPRO data structure

Return Value

none

Remarks

Should always be the last function called in the CHAPRO library.

See Also

cha_allocate

cha_data_gen
Generates C code that represents the CHAPRO data structure.

(int) **cha_data_gen**(CHA_PTR **cp**, char ***fn**)

Function arguments

cp	pointer to CHAPRO data structure
fn	Pointer to output filename.

Return Value

Error code:
0 – no error
1 – can't open output file
2 – data structure not yet initialized
3 – data structure contains no data

Remarks
The C code generated by this function represents the CHAPRO data structure after variables have been initialized and data memory has been allocated by prior calls to any preparation functions. The code is written to the file specified by **fn**.

See Also
cha_data_save, cha_data_load

cha_data_save

Writes the CHAPRO data structure to a binary file.

(int) **cha_data_save**(CHA_PTR **cp**, char ***fn**)

Function arguments

cp	pointer to CHAPRO data structure
fn	Pointer to output filename.

Return Value

Error code:
0 – no error
1 – can't open output file
2 – data structure not yet initialized
3 – data structure contains no data

Remarks

The binary file written by this function represents the CHAPRO data structure after variables have been initialized and data memory has been allocated by prior calls to any preparation functions. The code is written to the file specified by **fn**. A 16-byte file header is prepended to the data.

See Also

cha_data_gen, cha_data_load

cha_data_read
Reads the CHAPRO data structure from a binary file.

(int) **cha_data_read**(CHA_PTR **cp**, char ***fn**)

Function arguments

cp	pointer to CHAPRO data structure
fn	Pointer to output filename.

Return Value

Error code:
0 – no error
1 – can't open output file
2 – data structure not yet initialized
3 – data structure contains no data

Remarks
The binary file read by this function represents the CHAPRO data structure after variables have been initialized and data memory has been allocated by prior calls to any preparation functions. The code is written to the file specified by **fn**. A 16-byte file header is prepended to the data.

See Also

cha_data_save, cha_data_gen

cha_scale
Applies scales factor to a chuck of the input or output stream.

(void) **cha_scale**(float ***x**, int **cs**, float **scale**)

Function arguments

x	pointer to input signal
cs	chunk size
scale	pointer to CHAPRO data structure

Return Value

None.

Remarks

The scaled output signal overwrites the input signal.

cha_fft_cr

Inverse Fourier transform complex frequency components into real signal.

(void) **cha_fft_cr**(float ***x,** int **n**)

Function arguments

x	Complex frequency components are replaced by real-valued signal.
n	Number of points in the signal.

Return Value

None

Remarks

The input array must be dimensioned to accommodate **n+2** float values. The number of complex frequency components is **(n+2)/2.**

cha_fft_rc

Fourier transform real signal into complex frequency components.

(void) **cha_fft_rc**(float ***x,** int **n**)

Function arguments

x	Real-valued signal is replaced by complex frequency components
n	Number of points in the signal.

Return Value

None

Remarks

The input array must be dimensioned to accommodate **n+2** float values. The number of complex frequency components is **(n+2)/2.**

cha_fft
Fourier transform a complex time signal to complex frequency components.

(void) **cha_fft**(float ***x,** int **n**)

Function arguments

x	Complex frequency components are replaced by a complex signal.
n	Number of points in the signal.

Return Value

None

Remarks

The input array must be dimensioned to accommodate **n×2** float values.

cha_ifft
Inverse Fourier transform complex frequency components into a complex-valued signal.

(void) **cha_ifft**(float ***x,** int **n**)

Function arguments

x	Complex signal is replaced by complex frequency components
n	Number of points in the signal.

Return Value

None

Remarks
The input array must be dimensioned to accommodate **n×2** float values.

cha_prepare
CHAPRO data structure preparation function.

(void) **cha_prepare**(CHA_PTR **cp**)

Function arguments

cp	pointer to CHAPRO data structure

Return Value

None.

Remarks

Should be called only once and prior to calling other library functions; however, violations of this rule may be tolerated.

cha_version
Returns a string that describes the current version of the CHAPRO library.

(char *) **cha_version**(void)

Function arguments

none

Return Value

Pointer to version string.

Remarks

An example of the return value, "CHAPro version 0.03, 6-Nov-2016".

cha_agc_prepare
Automatic-gain-control preparation function.

(int) **cha_agc_prepare**(CHA_PTR **cp**, CHA_DSL ***dsl**, CHA_WDRC ***gha**)

Function arguments

cp	pointer to CHA data structure
dsl	pointer to DSL prescription structure (see Appendix C)
gha	pointer to WDRC prescription structure (see Appendix D)

Return Value

Error code:
0 – no error

Remarks

Initializes variables and allocates memory for automatic gain control. Chunk size is the number of samples read from the input signal and written to the output signal with each call to cha_agc_process.

See Also

cha_agc_process

cha_agc_input
Automatic-gain-control processing function.

(void) **cha_agc_input**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Performs single-channel, automatic-gain-control processing on CHAPRO input signal. Chunk size is the number of samples read from the input signal and written to the output signal.

See Also

cha_agc_prepare, cha_agc_output

cha_agc_channel
Automatic-gain-control processing function.

(void) **cha_agc_channel**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Performs multi-channel, automatic-gain-control processing. Chunk size is the number of samples read from the input signal and written to the output signal.

See Also

cha_agc_prepare

cha_agc_output
Automatic-gain-control processing function.

(void) **cha_agc_output**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Performs single-channel, automatic-gain-control processing on CHAPRO output signal. Chunk size is the number of samples read from the input signal and written to the output signal.

See Also

cha_agc_prepare, cha_agc_input

cha_icmp_prepare
Instantaneous-compression preparation function.

(int) **cha_icmp_prepare**(CHA_PTR **cp**,
float ***Lc**, float ***Gc**, double **lr**, int **np**, int **ds**)

Function arguments

cp	pointer to CHA data structure
Lc	pointer to level array
Lc	pointer to gain array
lr	level reference
np	number of points in level and gain arrays
ds	down-sample factor

Return Value

Error code:
0 – no error

Remarks

Initializes variables and allocates memory for instantaneous compression. Chunk size is the number of samples read from the input signal and written to the output signal with each call to cha_icmp_process.

See Also

cha_icmp_process

cha_icmp_process

Instantaneous-compression processing function.

(void) **cha_icmp_process**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Performs instantaneous compression. Chunk size is the number of samples read from the input signal and written to the output signal.

See Also

cha_icmp_prepare

cha_afc_prepare
Configure feedback management.

(int) **cha_afc_prepare**(CHA_PTR **cp**, CHA_AFC *afc**)**

Function arguments

cp	pointer to CHA data structure
afc	pointer to AFC parameters & buffers structure (see Appendix E)

Return Value

Error code:
0 – no error

Remarks

Initializes variables and allocates memory for adaptive feedback cancelation (AFC), which is controlled by contained in the CHA_AFC structure. In the current version (0.24), the AFC implementation is functional, but incomplete.

Feedback estimation is controlled by three parameters, **mu**, **rho**, & **eps**. For testing purposes, feedback simulation is enabled by setting **fbg**=1 or disabled by setting **fbg**=0. When simulation is enabled, the feedback-filter misalignment error is saved as a quality metric by setting **sqm**=1.

See Also

cha_afc_input, cha_afc_output

cha_afc_input
Process input signal to remove feedback.

(void) **cha_afc_input**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHAPRO data structure
x	pointer to input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Removes estimated feedback from input signal. Chunk size is the number of samples read from the input signal and written to the output signal. Optionally simulates feedback and saves misalignment error as a quality metric.

See Also

cha_afc_prepare, cha_afc_output

cha_afc_output
Save output signal for feedback management.

(void) **cha_afc_output**(CHA_PTR **cp**, float ***x**, int **cs**)

Function arguments

cp	pointer to CHAPRO data structure
x	pointer to input signal
cs	chunk size

Return Value
none

Remarks
Assists feedback management by saving the output of the hearing-aid processing. Chunk size is the number of samples read from the input signal.

See Also
cha_afc_prepare, cha_afc_input

cha_ciirfb_design
Complex IIR filter-bank design function.

(int) **cha_ciirfb_design**(float ***z**, float ***p**, float ***g**, int ***d**, int **nc**,
double ***fc**, double ***bw**, double **sr**, double **td**)

Function arguments

z	pointer to IIR filter complex zeros
p	pointer to IIR filter complex poles
g	pointer IIR filter complex gain
d	pointer IIR filter delay
nc	number of frequency bands
fc	pointer to list of center frequencies (Hz)
bw	pointer to list of bandwidths (Hz)
sr	sampling rate (samples/second)
td	target group delay (ms)

Return Value

Error code:
0 – no error

Remarks

Computes zeros and poles for the complex IIR filter-bank. The filterbank design is based on fourth-order gammatone bandpass filters. The center frequency (**fc**) and bandwidth (**bw**) arrays are input arguments and their size is equal to the number of frequency bands (**nc**). The zero (**z**) and pole (**p**) arrays are output arguments and are complex numbers, so are stored as sequential real and imaginary parts. The size of the zeros & poles arrays is two times the product of the number of frequency bands (**nc**) and the number of zeros (and poles) per band, which is always equal to 4. The gain array (**g**) is complex, so its size two times the number of frequency bands (**nc**). The size of the filter delay array (**d**) is equal to the number of frequency bands (**nc**). The filter-design process attempts to align the impulse response of each frequency band to the target delay (**td**).

See Also

cha_ciirfb_prepare, cha_ciirfb_analyze, cha_ciirfb_synthesize

cha_ciirfb_prepare
Complex IIR filter-bank preparation function.

(int) **cha_ciirfb_prepare**(CHA_PTR **cp**, float ***z**, float ***p**, float ***g**, int ***d**,
int **nc**, int **nz**, double **sr**, int **cs**)

Function arguments

cp	pointer to CHA data structure
z	pointer to IIR filter complex zeros
p	pointer to IIR filter complex poles
g	pointer IIR filter complex gain
d	pointer IIR filter delay
nc	number of frequency bands
nz	number of zeros (and poles) for each band
sr	sampling rate (samples/second)
cs	chunk size

Return Value

Error code:
0 – no error

Remarks

Initializes variables and allocates memory for the IIR filter-bank. The zero (**z**) and pole (**p**) arrays are output arguments and are complex numbers, so are stored as sequential real and imaginary parts. The size of the zero & pole arrays is two times the product of the number of frequency bands (**nc**) and the number of zeros per band (**nz**). The number of zeros (and poles) per band (**nz**) should be an even number and should include conjugate pairs, so that second-order sections will have real coefficients. The gain array (**g**) is complex, so its size two times the number of frequency bands (**nc**). The size of the filter delay array (**d**) is equal to the number of frequency bands (**nc**). Chunk size is the number of samples read from the input signal and written to the output signal.

See Also

cha_ciirfb_design, cha_ciirfb_analyze, cha_ciirfb_synthesize

cha_ciirfb_analyze

Complex IIR filter-bank frequency-analysis function.

(void) **cha_ciirfb_analyze**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to input signal
y	pointer to complex output signal
cs	chunk size

Return Value

none

Remarks

Performs complex IIR filter-bank analysis. Chunk size is the number of samples read from the input and written to the output.

See Also

cha_ciirfb_prepare, cha_ciirfb_synthesize

cha_ciirfb_synthesize

Complex IIR filter-bank frequency-synthesis function.

(void) **cha_ciirfb_synthesize**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to complex input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Performs complex filter-bank frequency-synthesis. Frequency bands are summed and only the real part is output. Chunk size is the number of samples read from the input and written to the output.

See Also

cha_ciirfb_prepare, cha_ciirfb_analyze

cha_cfirfb_prepare
Complex FIR filter-bank preparation function.

(int) **cha_cfirfb_prepare**(CHA_PTR **cp**, double ***cf**, int **nc**, double **sr**,
int **nw**, int **wt**, int **cs**)

Function arguments

cp	pointer to CHA data structure
cf	list frequency band edges (kHz)
nc	number of frequency bands
sr	sampling rate (samples/second)
nw	window size (samples)
wt	window type (0=Hamming, 1=Blackman)
cs	chunk size

Return Value

Error code:
0 – no error

Remarks

Initializes variables and allocates memory for the complex FIR filter-bank. Chunk size is the number of samples read from the input and written to the output.

See Also

cha_cfirfb_analyze, cha_cfirfb_synthesize

cha_cfirfb_analyze
Complex FIR filter-bank frequency-analysis function.

(void) **cha_cfirfb_analyze**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to real input signal
y	pointer to complex output signal
cs	chunk size

Return Value

none

Remarks

Performs complex FIR filter-bank analysis. Chunk size is the number of samples read from the input and written to the output.

See Also

cha_cfirfb_prepare, cha_cfirfb_synthesize

cha_cfirfb_synthesize
Complex FIR filter-bank frequency-synthesis function.

(void) **cha_cfirfb_synthesize**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to complex input signal
y	pointer to real output signal
cs	chunk size

Return Value

none

Remarks

Performs complex FIR filter-bank synthesis. Chunk size is the number of samples read from the input and written to the output.

See Also

cha_cfirfb_prepare, cha_cfirfb_analyze

cha_firfb_prepare
FIR filter-bank preparation function.

(int) **cha_firfb_prepare**(CHA_PTR **cp**,
double ***cf**, int **nc**, double **sr**, int **nw**, int **wt**, int **cs**)

Function arguments

cp	pointer to CHA data structure
cf	list frequency band edges (kHz)
nc	number of frequency bands
sr	sampling rate (samples/second)
nw	window size (samples)
wt	window type (0=Hamming, 1=Blackman)
cs	chunk size

Return Value

Error code:
0 – no error

Remarks

Initializes variables and allocates memory for the FIR filter-bank. Chunk size is the number of samples read from the input and written to the output.

See Also

cha_firfb_analyze, cha_firfb_synthesize

cha_firfb_analyze
FIR filter-bank frequency-analysis function.

(void) **cha_firfb_analyze**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Performs FIR filter-bank analysis. Chunk size is the number of samples read from the input signal and written to the output signal.

See Also

cha_firfb_prepare, cha_firfb_synthesize

cha_firfb_synthesize
FIR filter-bank frequency-synthesis function.

(void) **cha_firfb_synthesize**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Performs FIR filter-bank synthesis. Chunk size is the number of samples read from the input signal and written to the output signal.

cha_iirfb_design
IIR filter-bank design function.

(int) **cha_iirfb_design**(float ***z**, float ***p**, float ***g**, int ***d**,
double ***cf**, int **nc**, int **nz**, double **sr**, double **td**)

Function arguments

z	pointer to IIR filter complex zeros
p	pointer to IIR filter complex poles
g	pointer to IIR filter gain
d	pointer to IIR filter delay
cf	pointer to cross-over frequencies
nc	number of frequency bands
nz	number of zeros (and poles) for each band
sr	sampling rate (samples/second)
td	impulse response target delay (millisecond)

Return Value

Error code:
0 – no error

Remarks

Computes zeros and poles for the IIR filter-bank. The filterbank design is based on Butterworth bandpass filters sandwiched between Butterworth low-pass and high-pass filters. The array of cross-over frequencies (**cf**) is an input argument and its size is one less than the number of frequency bands (**nc**). The zero (**z**) and pole (**p**) arrays are output arguments and are complex numbers, so are stored as sequential real and imaginary parts. The size of the zeros & poles arrays is two times the product of the number of frequency bands (**nc**) and the number of zeros per band (**nz**). The number of zeros (and poles) per band (**nz**) should be an even number and should include conjugate pairs, so that second-order sections have real coefficients. The number of filter gains and filter delays (**g** & **d**) is the number of frequency bands (**nc**). The filter-design process attempts to align the impulse response of each frequency band to the target delay (**td**).

See Also

cha_iirfb_prepare, cha_iirfb_analyze, cha_iirfb_synthesize

cha_iirfb_prepare
IIR filter-bank preparation function.

(int) **cha_iirfb_prepare**(CHA_PTR **cp**, float ***z**, float ***p**, float ***g**, int ***d**,
int **nc**, int **nz**, double **sr**, int **cs**)

Function arguments

cp	pointer to CHA data structure
z	pointer to IIR filter complex zeros
p	pointer to IIR filter complex poles
g	pointer IIR filter gain
d	pointer IIR filter delay
nc	number of frequency bands
nz	number of zeros (and poles) for each band
sr	sampling rate (samples/second)
cs	chunk size

Return Value

Error code:
0 – no error

Remarks

Initializes variables and allocates memory for the IIR filter-bank. The zero (**z**) and pole (**p**) arrays are output arguments and are complex numbers, so are stored as sequential real and imaginary parts. The size of the zero & pole arrays is two times the product of the number of frequency bands (**nc**) and the number of zeros per band (**nz**). The number of zeros (and poles) per band (**nz**) should be an even number and should include conjugate pairs, so that second-order sections will have real coefficients. The filter gains and filter delays (**g** & **d**) are input arguments and have array sizes equal to the number of frequency bands (**nc**). Chunk size is the number of samples read from the input signal and written to the output signal.

See Also

cha_iirfb_design, cha_iirfb_analyze, cha_iirfb_synthesize

cha_iirfb_analyze

IIR filter-bank frequency-analysis function.

(void) **cha_iirfb_analyze**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Performs IIR filter-bank analysis. Chunk size is the number of samples read from the input signal and written to the output signal.

See Also

cha_iirfb_prepare, cha_iirfb_synthesize

cha_firfb_synthesize
IIR filter-bank frequency-synthesis function.

(void) **cha_iirfb_synthesize**(CHA_PTR **cp**, float ***x**, float ***y**, int **cs**)

Function arguments

cp	pointer to CHA data structure
x	pointer to input signal
y	pointer to output signal
cs	chunk size

Return Value

none

Remarks

Performs IIR filter-bank synthesis. Chunk size is the number of samples read from the input signal and written to the output signal.

See Also

cha_iirfb_prepare, cha_iirfb_analyze

Appendix A. Test programs

Several examples that test basic aspects of complex IIR filter-bank and instantaneous-compression.

- `tst_cifa` – tests complex IIR filter-bank analysis
- `tst_cifio` – tests simple waveform complex IIR processing
- `tst_cifsc` – tests simple waveform complex IIR processing with soundcard

Several examples that test basic aspects of FIR filter-bank and automatic-gain-control.

- `tst_ffa` – tests FIR filter-bank analysis
- `tst_ffio` – tests simple waveform FIR processing
- `tst_ffsc` – tests speech-waveform FIR & AGC with soundcard

Several examples that test basic aspects of IIR filter-bank with automatic-gain-control and adaptive feedback cancelation.

- `tst_ifa` – tests IIR filter-bank analysis
- `tst_ifio` – tests simple waveform IIR processing
- `tst_ifsc` – tests speech-waveform IIR & AGC
- `tst_gha` – tests speech-waveform IIR & AGC & AFC with soundcard

These test programs are all written in C and produce results that are written to a subfolder called “test.” Each test program has a corresponding MATLAB script for viewing the results. The test programs that contain IIR processing require filter coefficients to be precomputed by a MATLAB script called iirfb.

Appendix B. CLS Prescription

Structure CHA_CLS specifies the CLS prescription.

```
#define CLS_MXCH 32          // maximum number of channels

typedef struct {
    int cm;                  // compression mode
    int nc;                  // number of channels
    double fc[CLS_MXCH];     // center frequency
    double bw[CLS_MXCH];     // bandwith
    double Gcs[CLS_MXCH];    // gain at compression start
    double Gcm[CLS_MXCH];    // gain at compression middle
    double Gce[CLS_MXCH];    // gain at compression end
    double Gmx[CLS_MXCH];    // maximum gain
    double Lcs[CLS_MXCH];    // level at compression start
    double Lcm[CLS_MXCH];    // level at compression middle
    double Lce[CLS_MXCH];    // level at compression end
    double Lmx[CLS_MXCH];    // maximum output level
} CHA_CLS;
```

Appendix C. DSL Prescription

Structure CHA_DSL specifies the DSL prescription.

```
#define DSL_MXCH 32              // maximum number of channels

typedef struct {
    double attack;               // attack time (ms)
    double release;              // release time (ms)
    double maxdB;                // maximum output (dB SPL)
    int ear;                     // 0=left, 1=right
    int nchannel;                // number of channels
    double cross_freq[DSL_MXCH]; // cross frequencies (Hz)
    double tkgain[DSL_MXCH];     // compression-start gain
    double cr[DSL_MXCH];         // compression ratio
    double tk[DSL_MXCH];         // compression-start kneepoint
    double bolt[DSL_MXCH];       // broadband output limiting threshold
} CHA_DSL;
```

Appendix D. WDRC Parameters

Structure CHA_WDRC specifies single-channel WDRC parameters

```
typedef struct {
    double attack;              // attack time (ms)
    double release;             // release time (ms)
    double fs;                  // sampling rate (Hz)
    double maxdB;               // maximum signal (dB SPL)
    double tkgain;              // compression-start gain
    double tk;                  // compression-start kneepoint
    double cr;                  // compression ratio
    double bolt;                // broadband output limiting threshold
} CHA_WDRC;
```

Appendix E. AFC Parameters

Structure CHA_AFC specifies single-channel AFC parameters

```
typedef struct {
    // simulation parameters
    double fbg;                  // simulated-feedback gain
    // AFC parameters
    double rho;                  // forgetting factor
    double eps;                  // power threshold
    double  mu;                  // step size
    int    afl;                  // adaptive-filter length
    int    wfl;                  // whitening-filter length
    int    pfl;                  // persistent-filter length
    int    fbl;                  // simulated-feedback length
    int    hdel;                 // output/input hardware delay
    // feedback filter buffers
    float *efbp;                 // estimated-feedback buffer pointer
    float *sfbp;                 // simulated-feedback buffer pointer
    float *wfrp;                 // whitening-feedback buffer pointer
    float *ffrp;                 // persistent-feedback buffer pointer
    // quality metric buffers & parameters
    float *merr;                 // chunk-error buffer pointer
    float *qm;                   // quality-metric buffer pointer
    int    nqm;                  // quality-metric buffer size
    int    iqm;                  // quality-metric index
    int    sqm;                  // save quality metric ?
    CHA_PTR pcp;                 // previous CHA_PTR
} CHA_AFC;
```