libbpm Reference Manual

0.1

Generated by Doxygen 1.5.1

Fri Nov 9 21:17:10 2007

Contents

1 libbpm Main Page

Author:

Bino Maiheu, University College London
Mark Slater, University of Cambridge
Alexey Lyapin, University College London
Stewart Boogert, Royal Holloway University of London

1.1 Introduction

This is libbpm :)

1.2 compilation

1.2.1 under Linux/Unix/MacOS

Bla bla bla

1.2.2 on Compilation under Windows

To compile libbpm under windows, it is best to use the MinGW + MSYS environment which enables one to build native libraries under windows (dll). For this you need to declare some routines during the build process using the dllexport macro that MinGW defines. So when you want to compile this library as a DLL, set the BUILD_DLL define statement active below. Or compile using -DBUILD_DLL. When you want to use this headerfile to for linking with the bpm.ddl library, undefine the BUILD_DLL, this will enable the compiler to import routines from libbpm in other programs from the ddl. Under linux it does not make a difference as the if statement checks first for the existence of the DLL_EXPORT and __WIN32__ macros.

2 libbpm Module Index

2.1 libbpm Modules

Here is a list of all modules:

3 libbpm Data Structure Index

3.1 libbpm Data Structures

Here are the data structures with brief descriptions:

4 libbpm File Index

4.1 libbpm File List

Here is a list of all documented files with brief descriptions:

5 libbpm Module Documentation

5.1 Waveform memory allocation

5.1.1 Detailed Description

bpm_defs.h (p. ??)

Main definitions for libbpm as well as doxygen intro documentation

These are a number of definitions to make the code run on various systems (like e.g. win32...) and some other general definitions used by the library.

Files

- file **alloc_complex_wave_double.c**
- file **alloc_simple_wave_double.c**
- file **alloc_simple_wave_int.c**
- file **bpm_alloc.h**

  libbpm waveform memory allocation routines

Functions

- EXTERN double ** **alloc_complex_wave_double** (int ns)
- EXTERN void **free_complex_wave_double** (double **w, int ns)
- EXTERN double * **alloc_simple_wave_double** (int ns)
- EXTERN void **free_simple_wave_double** (double *w)
- EXTERN int * **alloc_simple_wave_int** (int ns)
- EXTERN void **free_simple_wave_int** (int *w)

5.1.2 Function Documentation

5.1.2.1 EXTERN double** alloc_complex_wave_double (int *ns*)

Allocates memory for a complex waveform of doubles. A pointer is returned to the reserved memory. Use free_complex_wave_double(w, ns) to free up the memory.

Parameters:

ns Number of samples

Returns:

a pointer to the reserved memory

Definition at line 8 of file alloc_complex_wave_double.c.

References bpm_error().

Referenced by ddc_sample_waveform(), ddc_waveform(), generate_diode(), generate_dipole(), generate_monopole(), rf_butterworthhighpass(), and rf_butterworthlowpass().

5.1.2.2 EXTERN void free_complex_wave_double (double ** *w*, int *ns*)

Frees up the memory reserved by alloc_complex_wave_double(ns).

Parameters:

w A pointer to a complex waveform of doubles

ns Number of samples

Returns:

void

Definition at line 34 of file alloc_complex_wave_double.c.

Referenced by ddc_sample_waveform(), ddc_waveform(), generate_diode(), generate_dipole(), generate_monopole(), rf_butterworthhighpass(), and rf_butterworthlowpass().

5.1.2.3 EXTERN double* alloc_simple_wave_double (int *ns*)

Allocates memory for a simple waveform of doubles. A pointer is returned to the reserved memory. Use free_simple_wave_double(w) to free up the memory.

Parameters:

ns Number of samples

Returns:

a pointer to the reserved memory

Definition at line 9 of file alloc_simple_wave_double.c.

References bpm_error().

Referenced by create_filter(), ddc_waveform(), fit_fft(), fit_waveform(), generate_dipole(), and generate_monopole().

5.1.2.4 EXTERN void free_simple_wave_double (double * *w*)

Frees up the memory reserved by alloc_simple_wave_double(ns).

Parameters:

w A pointer to a complex waveform of doubles

Returns:

void

Definition at line 27 of file alloc_simple_wave_double.c.

Referenced by ddc_waveform(), delete_filter(), fit_fft(), fit_waveform(), generate_dipole(), and generate_monopole().

5.1.2.5 EXTERN int* alloc_simple_wave_int (int *ns*)

Allocates memory for a simple waveform of integers. A pointer is returned to the reserved memory. Use free_simple_wave_int(w) to free up the memory.

Parameters:

ns Number of samples

Returns:

a pointer to the reserved memory

Definition at line 9 of file alloc_simple_wave_int.c.

References bpm_error().

Referenced by generate_diode(), generate_dipole(), and generate_monopole().

5.1.2.6 EXTERN void free_simple_wave_int (int * *w*)

Frees up the memory reserved by alloc_simple_wave_int(ns).

Parameters:

w A pointer to a complex waveform of integers

Returns:

void

Definition at line 26 of file alloc_simple_wave_int.c.

5.2 Analysis routines

Files

- file **ana_compute_residual.c**
- file **ana_def_cutfn.c**
- file **ana_get_svd_coeffs.c**
- file **ana_set_cutfn.c**
- file **bpm_analysis.h**

  libbpm analysis routines

Defines

- #define **BPM_GOOD_EVENT**
- #define **BPM_BAD_EVENT**
- #define **ANA_SVD_TILT**
- #define **ANA_SVD_NOTILT**

Functions

- EXTERN int **ana_set_cutfn** (int(*cutfn)(**bpmproc_t** *proc))
- EXTERN int **ana_get_svd_coeffs** (**bpmproc_t** **proc, int num_bpms, int num_svd, int total_num_evts, double *coeffs, int mode)
- EXTERN int **ana_compute_residual** (**bpmproc_t** **proc, int num_bpms, int num_evts, double *coeffs, int mode, double *mean, double *rms)
- EXTERN int **ana_def_cutfn** (**bpmproc_t** *proc)

Variables

- EXTERN int(*) **ana_cutfn** (**bpmproc_t** *proc)

5.2.1 Define Documentation

5.2.1.1 #define BPM_GOOD_EVENT

A good event

Definition at line 28 of file bpm_analysis.h.

Referenced by ana_compute_residual(), ana_def_cutfn(), ana_get_svd_coeffs(), and ana_set_cutfn().

5.2.1.2 #define BPM_BAD_EVENT

A bad event

Definition at line 29 of file bpm_analysis.h.

5.2.1.3 #define ANA_SVD_TILT

Include tilts in the SVD

Definition at line 31 of file bpm_analysis.h.

Referenced by ana_compute_residual(), and ana_get_svd_coeffs().

5.2.1.4 #define ANA_SVD_NOTILT

Don't include tilts in the SVD

Definition at line 32 of file bpm_analysis.h.

5.2.2 Function Documentation

5.2.2.1 EXTERN int ana_set_cutfn (int(*)(bpmproc_t *proc) *cutfn*)

Set the cut function

Parameters:

cutfn a pointer to the cut function with a bpmproc_t as argument

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file ana_set_cutfn.c.

References ana_cutfn, bpm_error(), and BPM_GOOD_EVENT.

5.2.2.2 EXTERN int ana_get_svd_coeffs (bpmproc_t ** *proc*, int *num_bpms*, int *num_svd*, int *total_num_evts*, double * *coeffs*, int *mode*)

Perform the SVD on the given data and return the coefficients. The index 0 bpmconf is the bpm to be regressed against and the remainder are put into the regression. The coeffs array must be valid up to the number of arguments approapriate to mode.

Parameters:

proc pointer to the the processed bpm databuffer

num_bpms the number of bpms in the array

num_svd number of svd constants

total_num_evts total number of events in the buffer

coeffs the array of correlation coefficients that is returned

mode mode option: take tilts into account in the SVD ?

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 9 of file ana_get_svd_coeffs.c.

References ana_cutfn, ANA_SVD_TILT, BPM_GOOD_EVENT, gsl_linalg_SV_decomp(), gsl_linalg_SV_solve(), gsl_matrix_calloc(), gsl_matrix_set(), gsl_vector_calloc(), gsl_vector_get(), and gsl_vector_set().

5.2.2.3 EXTERN int ana_compute_residual (bpmproc_t ** *proc*, int *num_bpms*, int *num_evts*, double * *coeffs*, int *mode*, double * *mean*, double * *rms*)

Calculate the mean and rms of the residual fomr the given events. Note that the mode and svd coefficients must 'match' as with **ana_get_svd_coeffs()** (p. 12)

Parameters:

proc pointer to the the processed bpm databuffer

num_bpms the number of bpms in the array

num_evts total number of events in the buffer

coeffs the array of correlation coefficients

mode mode option: take tilts into account in the SVD ?

mean the returned mean

rms the returned rms

Definition at line 8 of file ana_compute_residual.c.

References ana_cutfn, ANA_SVD_TILT, BPM_GOOD_EVENT, and bpmproc::ddc_pos.

5.2.2.4 EXTERN int ana_def_cutfn (bpmproc_t * *proc*)

The default cut function if people cut be bothered to do their own :)

Parameters:

proc the event to decide

Returns:

BPM_GOOD_EVENT if the event is good, BPM_BAD_EVENT if it isn't

Definition at line 10 of file ana_def_cutfn.c.

References BPM_GOOD_EVENT.

5.2.3 Variable Documentation

5.2.3.1 EXTERN int(*) ana_cutfn(bpmproc_t *proc)

A user cut function to allow cuts to be applied while selecting events for SVD, etc.

Definition at line 100 of file bpm_analysis.h.

5.3 Calibration routines

Files

- file **bpm_calibration.h**
  calibration routines
- file **calibrate.c**
- file **calibrate_svd.c**
- file **load_calibration.c**
- file **save_calibration.c**
- file **setup_calibration.c**
- file **update_freq_tdecay.c**

Functions

- EXTERN int **setup_calibration** (**bpmconf_t** *cnf, **bpmproc_t** *proc, int npulses, int startpulse, int stoppulse, double angle, double startpos, double endpos, int num_steps, **beamconf_t** *beam)
- EXTERN int **calibrate** (**bpmconf_t** *bpm, **beamconf_t** *beam, **bpmproc_t** *proc, int npulses, **bpmcalib_t** *cal)
- EXTERN int **update_freq_tdecay** (**bpmproc_t** *proc, int npulses, **bpmcalib_t** *cal)
- EXTERN int **calibrate_svd** (**beamconf_t** **beam, **bpmconf_t** **bpm, **bpmproc_t** **proc, int npulses, int nbpms, int *bpmidx, **bpmcalib_t** *cal)
- EXTERN int **save_calibration** (char *fname, **bpmconf_t** *bpm, **bpmcalib_t** *cal, int num_bpms)
- EXTERN int **load_calibration** (char *fname, **bpmconf_t** *bpm, **bpmcalib_t** *cal, int num_bpms)

5.3.1 Function Documentation

5.3.1.1 EXTERN int setup_calibration (bpmconf_t * *cnf*, bpmproc_t * *proc*, int *npulses*, int *startpulse*, int *stoppulse*, double *angle*, double *startpos*, double *endpos*, int *num_steps*, beamconf_t * *beam*)

This routine basically defines the calibration steps and returns them into the array of beam structures. It needs an array of processed waveform structures, of dimension npulses from a single BPM. From this it determines the corresponding corrector/mover steps and puts them back into the array of beam structures given the bpm configurations.

Startpulse and stoppulse have to be in the first and last calib steps & will need some extensive error checking for e.g. missed calibration steps...

NOTE: This is not definitive yet - more checking, etc. required!

- DDC or FIT?
- Sign errors?
- not robust to missing steps

Parameters:

proc array of processed waveforms for a single bpm, so array of pulses

cnf array of bpm configuration structures

npulses number of pulses in the calibration

startpulse start of calibration range

stoppulse stop of calibration range

angle

startpos start position of calibration

endpos end position of calibration

num_steps number of calibration steps

beam the returned beamconf array which represents where the beam is supposed to be in each bpm during each calibration step

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file setup_calibration.c.

References bpm_error(), and beamconf::bpmhit.

5.3.1.2 EXTERN int calibrate (bpmconf_t * *bpm*, beamconf_t * *beam*, bpmproc_t * *proc*, int *npulses*, bpmcalib_t * *cal*)

Gets the calibration constants from an array of npulses of beam positions and processed waveform structures and returns an updated calibration structure. Note that this routine updates the IQ phase, the position scale and the tilt scale but DOES NOT touch the frequency, decay time or the t0Offset.

Parameters:

bpm Bpm structures

beam An array of beam structures, one for each pulse, so essentially this corresponds to where we expect the beam to be in each pulse, so representing corrector positions or mover positions. This information should be filled by the routine setup_calibration(...)

proc An array of processed waveforms, one for each pulse, which correspond to calculated positions that were calculated using IQ phase = 0 and scales equal to 1.

npulses The number of pulses in the arrays

∗***cal*** The returned calibration structure for the BPM that was calibrated

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 9 of file calibrate.c.

References bpm_error(), beamconf::bpmhit, bpmconf::cav_polarisation, bpmproc::ddc_Q, get_pos(), horiz, bpmcalib::IQphase, and nr_fit().

5.3.1.3 EXTERN int update_freq_tdecay (bpmproc_t ∗ *proc*, int *npulses*, bpmcalib_t ∗ *cal*)

Gets the list of processed pulses and refills the calib structure with updated frequencies and decay constants

NOT IMPLEMENTED YET !

Parameters:

proc array of processed waveforms

npulses the number of pulses

cal the refilled calibration structure

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 8 of file update_freq_tdecay.c.

References bpm_error().

5.3.1.4 EXTERN int calibrate_svd (beamconf_t ∗∗ *beam*, bpmconf_t ∗∗ *bpm*, bpmproc_t ∗∗ *proc*, int *npulses*, int *nbpms*, int ∗ *bpmidx*, bpmcalib_t ∗ *cal*)

The 2D arrays in this routine represent a set of collected pulses for all the bpms, so having beam[iBPM][iPulse], cnf[iBPM][iPulse] and proc[iBPM][iPulse],

Parameters:

npulses The number of pulses collected for calibration

Used for mover calibrations with at least 2 spectator bpms. eats something of the sort bpms[bpmidx][pulseidx], for a number of pulse and. nbpms specifies the total number of bpms involved in the regression bpmidx specifies the indexes of the bpms involve in the regression, bpmidx[0] gives the central bpm, for which the calibration is calculated the rest (bpmidx[1] -> bpmidx[nbpms-1]) gives the indexes of the spectator bpms.

NOT IMPLEMENTED YET !

Parameters:

beam

bpm

proc

npulses

nbpms The total number of BPMs that will be used in the regression

bpmidx An array of bpm indexes, where bpmidx[0] corresponds to the index of the bpm in the main array that we will calibrate, and the rest of the indices corresponds to the BPMs we will regress against, so basically the spectator BPMs, when doing a mover calibration

cal This structure is filled with the calculated iqphase, and position and resolution scales

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file calibrate_svd.c.

References bpm_error().

5.3.1.5 EXTERN int save_calibration (char * *fname*, bpmconf_t * *bpm*, bpmcalib_t * *cal*, int *num_bpms*)

Save the given calibrations with the given filename.

Parameters:

fname The filename to save as

bpm BPM configs - to provide a name and index

cal The calibrations to save

num_bpms The number of bpm cals to save

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 8 of file save_calibration.c.

References bpm_error().

5.3.1.6 EXTERN int load_calibration (char * *fname*, bpmconf_t * *bpm*, bpmcalib_t * *cal*, int *num_bpms*)

Load the calibration from the given filename.

Parameters:

fname The filename to load#

bpm BPM configs - to provide a name and index

cal The calibrations to load

num_bpms The number of bpm cals to load

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 8 of file load_calibration.c.

References bpm_error(), bpmcalib::ddcepsFilt, bpmcalib::ddcfiltBW, bpmcalib::freq, bpmcalib::IQphase, bpmcalib::posscale, bpmcalib::slopescale, bpmcalib::t0Offset, and bpmcalib::tdecay.

5.4 Digital signal processing routines

Files

- file **bpm_dsp.h**

  libbpm digital signal processing routines

- file **calculate_filter_coefficients.c**
- file **create_filter.c**
- file **create_resonator_representation.c**
- file **create_splane_representation.c**
- file **delete_filter.c**
- file **filter_impulse_response.c**
- file **filter_step_response.c**
- file **normalise_filter.c**
- file **print_filter.c**
- file **print_filter_representation.c**
- file **zplane_transform.c**

Data Structures

- struct **filterrep_t**
- struct **filter_t**

Defines

- #define **BESSEL**
- #define **BUTTERWORTH**
- #define **CHEBYSHEV**
- #define **RAISEDCOSINE**
- #define **RESONATOR**
- #define **GAUSSIAN**
- #define **BILINEAR_Z_TRANSFORM**
- #define **MATCHED_Z_TRANSFORM**
- #define **NO_PREWARP**
- #define **LOWPASS**
- #define **HIGHPASS**
- #define **BANDPASS**
- #define **BANDSTOP**
- #define **NOTCH**
- #define **ALLPASS**

- #define **MAXORDER**
- #define **MAXPZ**
- #define **FILT_EPS**
- #define **MAX_RESONATOR_ITER**

Functions

- EXTERN **filter_t** * **create_filter** (char name[], unsigned int options, int order, int ns, double fs, double f1, double f2, double par)
- EXTERN int **apply_filter** (**filter_t** *f, double *wf)
- EXTERN void **print_filter** (FILE *of, **filter_t** *f)
- EXTERN void **delete_filter** (**filter_t** *f)
- EXTERN int **filter_step_response** (**filter_t** *f, double *wf, int itrig)
- EXTERN int **filter_impulse_response** (**filter_t** *f, double *wf, int itrig)
- EXTERN **filterrep_t** * **create_splane_representation** (**filter_t** *f)
- EXTERN **filterrep_t** * **create_resonator_representation** (**filter_t** *f)
- EXTERN **filterrep_t** * **zplane_transform** (**filter_t** *f, **filterrep_t** *s)
- EXTERN void **print_filter_representation** (FILE *of, **filterrep_t** *r)
- EXTERN int **normalise_filter** (**filter_t** *f, **filterrep_t** *s)
- EXTERN int **calculate_filter_coefficients** (**filter_t** *f)
- EXTERN int **_expand_complex_polynomial** (**complex_t** *w, int n, **complex_t** *a)
- EXTERN **complex_t** **_eval_complex_polynomial** (**complex_t** *a, int n, **complex_t** z)

5.4.1 Define Documentation

5.4.1.1 #define BESSEL

Bitmask for Bessel filter

Definition at line 32 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

5.4.1.2 #define BUTTERWORTH

Bitmask for Butterworth filter

Definition at line 33 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

5.4.1.3 #define CHEBYSHEV

Bitmask for Chebyshev filter

Definition at line 34 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

5.4.1.4 #define RAISEDCOSINE

Bitmask for Raised Cosine filter

Definition at line 35 of file bpm_dsp.h.

5.4.1.5 #define RESONATOR

Bitmask for Resonator filter

Definition at line 36 of file bpm_dsp.h.

Referenced by create_filter().

5.4.1.6 #define GAUSSIAN

Bitmask for Gaussian filter

Definition at line 37 of file bpm_dsp.h.

5.4.1.7 #define BILINEAR_Z_TRANSFORM

Get z poles via bilinear z transform from s plane

Definition at line 39 of file bpm_dsp.h.

5.4.1.8 #define MATCHED_Z_TRANSFORM

Get z poles via matches z transform from s plane

Definition at line 40 of file bpm_dsp.h.

Referenced by zplane_transform().

5.4.1.9 #define NO_PREWARP

Don't do the prewarp correction

Definition at line 41 of file bpm_dsp.h.

Referenced by create_filter().

5.4.1.10 #define LOWPASS

Normalise filter as lowpass

Definition at line 43 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and normalise_filter().

5.4.1.11 #define HIGHPASS

Normalise filter as highpass

Definition at line 44 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and normalise_filter().

5.4.1.12 #define BANDPASS

Normalise filter as bandpass

Definition at line 45 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients().

5.4.1.13 #define BANDSTOP

Normalise filter as bandstop

Definition at line 46 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and create_resonator_representation().

5.4.1.14 #define NOTCH

Normalise filter as notch filter (=bandstop)

Definition at line 47 of file bpm_dsp.h.

5.4.1.15 #define ALLPASS

Normalise filter as allpass (resonator)

Definition at line 48 of file bpm_dsp.h.

Referenced by create_resonator_representation().

5.4.1.16 #define MAXORDER

Maximum filter order

Definition at line 50 of file bpm_dsp.h.

5.4.1.17 #define MAXPZ

Maximum number of poles and zeros

Definition at line 51 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and create_resonator_representation().

5.4.1.18 #define FILT_EPS

A small number used in bpmdsp

Definition at line 52 of file bpm_dsp.h.

Referenced by _expand_complex_polynomial(), create_resonator_representation(), and print_filter().

5.4.1.19 #define MAX_RESONATOR_ITER

Maximum iterations in resonator poles calculation

Definition at line 53 of file bpm_dsp.h.

Referenced by create_resonator_representation().

5.4.2 Function Documentation

5.4.2.1 EXTERN filter_t* create_filter (char *name*[ ], unsigned int *options*, int *order*, int *ns*, double *fs*, double *f1*, double *f2*, double *par*)

Creates the filter.

Parameters:

name a name for the filter

options filter specification and options bitword

order filter order

ns number of samples of the waveforms

fs sampling frequency

f1 first frequency

f2 optional second frequency (bandpass/bandstop)

par optional parameter

- for chebyshev : ripple in dB
- for resonator : Q factor

Returns:

A pointer to the created filter structure, memory is allocated on the heap inside this routine, the user has to take of deleting it using **delete_filter()** (p. 22).

Definition at line 10 of file create_filter.c.

References alloc_simple_wave_double(), filter_t::alpha1, filter_t::alpha2, BESSEL, bpm_error(), bpm_warning(), BUTTERWORTH, calculate_filter_coefficients(), filter_t::cheb_ripple, CHEBYSHEV, filter_t::cplane, create_resonator_representation(), create_splane_representation(), filter_t::f1, filter_t::f2, filter_t::fs, filter_t::IsFIR, filter_t::name, NO_PREWARP, normalise_filter(), filterrep_t::npoles, filter_t::ns, filter_t::options, filter_t::order, filter_t::Q, RESONATOR, filter_t::w_alpha1, filter_t::w_alpha2, filter_t::wfbuffer, filter_t::yc, and zplane_transform().

5.4.2.2 EXTERN int apply_filter (filter_t * *f*, double * *wf*)

Apply the filter to the given waveform. Note that the filter is applied in place, the user has to make a copy of the waveform if he/she wants to keep the original before applying the filter. The number of samples in the waveform has to be set in advance when creating the filter, it is stored in the filter structure (f->ns).

Parameters:

f pointer to a filter that was created using create_filter

wf an array containing the waveform to be filtered

Returns:

BPM_SUCCESS upon success and BPM_FAILURE upon failure

Definition at line 19 of file apply_filter.c.

References bpm_error(), filter_t::gain, filter_t::IsFIR, filter_t::ns, filter_t::nxc, filter_t::nyc, filter_t::wfbuffer, filter_t::xc, filter_t::xv, filter_t::yc, and filter_t::yv.

Referenced by filter_impulse_response(), and filter_step_response().

5.4.2.3 EXTERN void print_filter (FILE * *of*, filter_t * *f*)

Prints the filter to the given file pointer.

Parameters:

of the filepointer, use "stdout" to print to the terminal

f the filter to be printed

Returns:

void

Definition at line 8 of file print_filter.c.

References bpm_error(), c_abs(), c_arg(), filter_t::cplane, filter_t::dc_gain, filter_t::fc_gain, FILT_EPS, filter_t::hf_gain, filter_t::name, filter_t::nxc, print_filter_representation(), and filter_t::xc.

5.4.2.4 EXTERN void delete_filter (filter_t * *f*)

Clears the memory that was allocated on the heap for the filter f.

Parameters:

f a pointer to the filter

Returns:

void

Definition at line 8 of file delete_filter.c.

References filter_t::cplane, free_simple_wave_double(), and filter_t::wfbuffer.

5.4.2.5 EXTERN int filter_step_response (filter_t * *f*, double * *wf*, int *itrig*)

This routine fills the given wf with the step response of the filter. The step response is defined as wf[i] = 0. for i < itrig and wf[i] = 1. for i >= itrig.

Parameters:

f a pointer to the filter to use

wf pointer to a waveform which will be overwritten with the step response

itrig the sample number in the waveform which will have the step

Returns:

BPM_SUCCESS upon succes and BPM_FAILURE upon failure

Produces a stepresponse for the filter, step is defined by the trigger sample number the starting level and the endlevel

Definition at line 8 of file filter_step_response.c.

References apply_filter(), bpm_error(), and filter_t::ns.

5.4.2.6 EXTERN int filter_impulse_response (filter_t * *f*, double * *wf*, int *itrig*)

This routine fills the given wf with the impulse response of the filter. The impulse response is defined as wf[i] = 1. for i == itrig and wf[i] = 0. elsewhere.

Parameters:

f a pointer to the filter to use

wf pointer to a waveform which will be overwritten with the impulse response

itrig the sample number in the waveform which will have the impulse

Returns:

BPM_SUCCESS upon succes and BPM_FAILURE upon failure

Produces an impulse response for the filter, step is defined by the trigger sample number the starting level and the endlevel

Definition at line 7 of file filter_impulse_response.c.

References apply_filter(), bpm_error(), and filter_t::ns.

5.4.2.7 EXTERN filterrep_t* create_splane_representation (filter_t * *f*)

This routine returns a pointer to a filter representation **filterrep_t** (p. 93) in the s plane for Butterworth, Chebyshev and Bessel filters. It need an initialised filter structure which has the filter type and the order set. Memory is allocated for this routine on the heap, so the user is responsible to delete this memory using free().

Parameters:

f the initialised filter with the correct options in f->options

Returns:

the filter representation in the s plane

Definition at line 32 of file create_splane_representation.c.

References _add_splane_pole(), BESSEL, bpm_error(), BUTTERWORTH, c_conj(), c_exp(), filter_t::cheb_ripple, CHEBYSHEV, complex(), filterrep_t::npoles, filter_t::options, and filter_t::order.

Referenced by create_filter().

5.4.2.8 EXTERN filterrep_t* create_resonator_representation (filter_t * *f*)

This routine returns a pointer to a filter representation **filterrep_t** (p. 93) in the z plane for resonance filters. It needs an initialised filter structure which has the filter type and the Q factor set. Memory is allocated for this routine on the heap, so the user is responsible to delete this memory using free().

Parameters:

f the initialised filter with the correct options in f->options

Returns:

the filter representation in the z plane

Definition at line 15 of file create_resonator_representation.c.

References _eval_complex_polynomial(), _expand_complex_polynomial(), _reflect(), ALLPASS, filter_t::alpha1, BANDSTOP, bpm_error(), c_conj(), c_div(), c_exp(), complex(), FILT_EPS, complex_t::im, MAX_RESONATOR_ITER, MAXPZ, filterrep_t::npoles, filterrep_t::nzeros, filter_t::options, filterrep_t::pole, filter_t::Q, complex_t::re, and filterrep_t::zero.

Referenced by create_filter().

5.4.2.9 EXTERN filterrep_t* zplane_transform (filter_t * *f*, filterrep_t * *s*)

This routine transforms the poles and zeros for Bessel, Chebyshev and Butterworth filters to the z plane either via matched z transform or bilinear z transform. This is set in f->options. Memory is allocated for this routine on the heap, so the user is responsible to delete this memory using free().

Parameters:

f the filter, needs the options from it to check how to transform

s filter s plane poles and zeros

Returns:

a pointer to the z plane representation

Definition at line 8 of file zplane_transform.c.

References bpm_error(), c_div(), c_exp(), c_scale(), c_sum(), complex(), MATCHED_Z_TRANSFORM, filterrep_t::npoles, filterrep_t::nzeros, filter_t::options, filterrep_t::pole, and filterrep_t::zero.

Referenced by create_filter().

5.4.2.10 EXTERN void print_filter_representation (FILE * *of*, filterrep_t * *r*)

Prints the filter representation in terms of poles and zeros to the filepointer.

Parameters:

of the filepointer, use "stdout" to print to the terminal

r the filter representation to be printed

Returns:

void

Display filter representation

Definition at line 8 of file print_filter_representation.c.

References c_imag(), c_real(), filterrep_t::npoles, filterrep_t::nzeros, filterrep_t::pole, and filterrep_t::zero.

Referenced by print_filter().

5.4.2.11 EXTERN int normalise_filter (filter_t * *f*, filterrep_t * *s*)

Normalises the Butterworth, Chebyshev or Bessel filters to be Bandpass/stop or Low/Highpass

Parameters:

f the filter

s the filter's representation in the s plane

Returns:

BPM_SUCCESS upon success or BPM_FAILURE upon failure.

Definition at line 7 of file normalise_filter.c.

References bpm_error(), c_div(), c_scale(), complex(), HIGHPASS, LOWPASS, filterrep_t::npoles, filterrep_t::nzeros, filter_t::options, filterrep_t::pole, filter_t::w_alpha1, filter_t::w_alpha2, and filterrep_t::zero.

Referenced by create_filter().

5.4.2.12 EXTERN int calculate_filter_coefficients (filter_t * *f*)

Calculates the filter coefficients from the z plane representation. Before this routine is called, one has to make sure that the member cplane, which holds a pointer to the filter's representation in the complex plane is set. This routine than calculates the filter coefficients and stores them in f->xc (coefficients of x[n], x[n-1], x[n-2]...) and f->yc (coefficients of y[n-1], y[n-2], y[n-3], ... in case of IIR filters).

Parameters:

f the filter, having it's f->cplane member set to the z plan representation

Returns:

BPM_SUCCESS upon success or BPM_FAILURE upon failure.

Calculates the filter coefficients from the poles and zeros in the cplane representation... Also calculates the filter gains...

Definition at line 56 of file calculate_filter_coefficients.c.

References _eval_complex_polynomial(), _expand_complex_polynomial(), filter_t::alpha1, filter_t::alpha2, BANDPASS, BANDSTOP, c_abs(), c_div(), c_mult(), c_real(), c_sqrt(), complex(), filter_t::cplane, filter_t::dc_gain, filter_t::fc_gain, filter_t::gain, filter_t::hf_gain, HIGHPASS, LOWPASS, MAXPZ, filterrep_t::npoles, filter_t::nxc, filter_t::nyc, filterrep_t::nzeros, filter_t::options, filterrep_t::pole, filter_t::xc, filter_t::yc, and filterrep_t::zero.

Referenced by create_filter().

5.4.2.13 EXTERN int _expand_complex_polynomial (complex_t * *w*, int *n*, complex_t * *a*)

Helper routine to expand a complex polynomial from a set of zeros.

Parameters:

w array of complex zeros for the polynomial

n nunber of zeros

a array of coeffiecients for the polynomial that is returned

Returns:

BPM_SUCCESS upon success or BPM_FAILURE upon failure.

Calculate the polynomial coefficients in a0 + a1 * z + a2 * z^2 + a3 * z^3 + ... = (z-w1)(z-w2)(z-w3)... from the n polynomial's zero's "w" returns the results in a, the array of coefficients...

Definition at line 8 of file calculate_filter_coefficients.c.

References bpm_error(), c_imag(), c_mult(), c_neg(), c_sum(), complex(), and FILT_EPS.

Referenced by calculate_filter_coefficients(), and create_resonator_representation().

5.4.2.14 EXTERN complex_t _eval_complex_polynomial (complex_t * *a*, int *n*, complex_t *z*)

Helper routine to evaluate a complex polynomial for value z

Parameters:

a array of coeffiecients for the polynomial that is returned

n number of zeros

z the value for which to evalute the polynomial

Returns:

the value of the polynomial for z (**complex_t** (p. 88))

Definition at line 44 of file calculate_filter_coefficients.c.

References c_mult(), c_sum(), and complex().

Referenced by calculate_filter_coefficients(), and create_resonator_representation().

5.5 Beam orbit generation

Files

- file **bpm_orbit.h**
  libbpm orbit generation routines
- file **generate_bpm_orbit.c**
- file **generate_corr_scan.c**
- file **generate_mover_scan.c**
- file **vm.c**

Data Structures

- struct **v3**
- struct **m33**

Functions

- EXTERN int **generate_bpm_orbit** (**beamconf_t** *beam, **bpmconf_t** *bpm)
- EXTERN int **generate_corr_scan** (**bpmconf_t** *bpm, **beamconf_t** *beam, int num_evts, int num_steps, double angle_range, double angle, double z_pos)
- EXTERN int **generate_mover_scan** (**beamconf_t** *beam, int num_evts, int num_steps, double mover_range, double angle)
- void **v_copy** (struct **v3** *v1, struct **v3** *v2)
- double **v_mag** (struct **v3** *v1)
- void **v_scale** (struct **v3** *v1, double dscale)
- void **v_norm** (struct **v3** *v1)
- void **v_matmult** (struct **m33** *m1, struct **v3** *v1)
- void **v_add** (struct **v3** *v1, struct **v3** *v2)
- void **v_sub** (struct **v3** *v1, struct **v3** *v2)
- double **v_dot** (struct **v3** *v1, struct **v3** *v2)
- void **v_cross** (struct **v3** *v1, struct **v3** *v2)
- void **v_print** (struct **v3** *v1)
- void **m_rotmat** (struct **m33** *m1, double alpha, double beta, double gamma)
- void **m_matmult** (struct **m33** *m, struct **m33** *m1, struct **m33** *m2)
- void **m_matadd** (struct **m33** *m1, struct **m33** *m2)
- void **m_print** (struct **m33** *m1)

5.5.1 Function Documentation

5.5.1.1 EXTERN int generate_bpm_orbit (beamconf_t * *beam*, bpmconf_t * *bpm*)

Generate the beam at the bpm position, so takes the coordinates from the bpm in the global from (stored in bpm->geom_pos and bpm->geom_tilt and fills the local hit positions for the beam in the bpm, beam->bpmhit... Also transports the energy, charge etc.. through to this point...

- generates the beam at bpm position
- transports the energy, charge
- sets the bunch arrival time in each cavity, offsetted by digi_trigtimeoffset in the bpmconf

Parameters:

beam the beam configuration

bpm the bpm confguration

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file generate_bpm_orbit.c.

References beamconf::beampos, beamconf::beamslope, bpm_error(), beamconf::bpmhit, beamconf::bpmtilt, bpmconf::geom_pos, bpmconf::geom_tilt, m_rotmat(), v_add(), v_copy(), v_cross(), v_dot(), v_matmult(), v_scale(), v_sub(), v3::x, v3::y, and v3::z.

5.5.1.2 EXTERN int generate_corr_scan (bpmconf_t * *bpm*, beamconf_t * *beam*, int *num_evts*, int *num_steps*, double *angle_range*, double *angle*, double *z_pos*)

Fill the beamconf structures with the lab coords of a corrector scan

Parameters:

bpm A bpmconf structure containing the info about the BPM

beam The beamconf structure that contains the beam info

num_evts The number of events in each corrector scan step

num_steps The number of corrector scan steps

angle_range The angle over which the corrector scan is performed (in urad)

angle The orientation (from the horizontal) of the corrector scan axis (in urad)

z_pos The z position in the beamline of the corrector

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file generate_corr_scan.c.

References beamconf::beampos, beamconf::beamslope, bpm_error(), bpm_warning(), and bpmconf::geom_pos.

5.5.1.3 EXTERN int generate_mover_scan (beamconf_t * *beam*, int *num_evts*, int *num_steps*, double *mover_range*, double *angle*)

Fill the beamconf structures with the lab coords of a mover scan. At present, this just changes the beam coords rather than the physical bpm coords. In the future, should add the possiblility of time-varying BPM positions

Parameters:

beam The beamconf structure that contains the beam info

num_evts The number of events in each corrector scan step

num_steps The number of corrector scan steps

mover_range The size of the move (in um)

angle The orientation (from the horizontal) of the mover scan axis (in urad)

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file generate_mover_scan.c.

References beamconf::beampos, bpm_error(), and bpm_warning().

5.5.1.4 void v_copy (struct v3 * *v1*, struct v3 * *v2*)

Copy 3-vector v2 into 3-vector v1

Definition at line 11 of file vm.c.

References v3::x, v3::y, and v3::z.

Referenced by generate_bpm_orbit().

5.5.1.5 double v_mag (struct v3 * *v1*)

Return the magnitude of 3-vector v1

Definition at line 18 of file vm.c.

References v_dot().

Referenced by v_norm().

5.5.1.6 void v_scale (struct v3 * *v1*, double *dscale*)

Scale 3-vector v1 with factor dscale

Definition at line 22 of file vm.c.

References v3::x, v3::y, and v3::z.

Referenced by generate_bpm_orbit(), and v_norm().

5.5.1.7 void v_norm (struct v3 * *v1*)

Normalise 3-vector v1 to unit vector

Definition at line 28 of file vm.c.

References v_mag(), and v_scale().

5.5.1.8 void v_matmult (struct m33 * *m1*, struct v3 * *v1*)

Multiply matrix m1 with 3-vector v1 : m1.v1, result is in v1

Definition at line 32 of file vm.c.

References m33::e, v3::x, v3::y, and v3::z.

Referenced by generate_bpm_orbit().

5.5.1.9 void v_add (struct v3 * *v1*, struct v3 * *v2*)

Add two 3-vectors v1 and v2, result is in v1

Definition at line 44 of file vm.c.

References v3::x, v3::y, and v3::z.

Referenced by generate_bpm_orbit().

5.5.1.10 void v_sub (struct v3 * *v1*, struct v3 * *v2*)

Subtract 3-vectors v1 - v2, result is in v1

Definition at line 50 of file vm.c.

References v3::x, v3::y, and v3::z.

Referenced by generate_bpm_orbit().

5.5.1.11 double v_dot (struct v3 * *v1*, struct v3 * *v2*)

Return Scalar product of 3-vectors v1 and v2

Definition at line 56 of file vm.c.

References v3::x, v3::y, and v3::z.

Referenced by generate_bpm_orbit(), and v_mag().

5.5.1.12 void v_cross (struct v3 * *v1*, struct v3 * *v2*)

Return the vector product of 3 vectors v1 x v2, result is in v1

Definition at line 60 of file vm.c.

References v3::x, v3::y, and v3::z.

Referenced by generate_bpm_orbit().

5.5.1.13 void v_print (struct v3 * *v1*)

Print the 3-vector to stdout

Definition at line 74 of file vm.c.

References v3::x, v3::y, and v3::z.

5.5.1.14 void m_rotmat (struct m33 * *m1*, double *alpha*, double *beta*, double *gamma*)

Create rotation 3x3 matrix with the 3 euler angles alpha, beta and gamma, result in m1

Definition at line 78 of file vm.c.

References m33::e, and m_matmult().

Referenced by generate_bpm_orbit().

5.5.1.15 void m_matmult (struct m33 * *m*, struct m33 * *m1*, struct m33 * *m2*)

3x3 Matrix multiplication m1.m2, result in m

Definition at line 126 of file vm.c.

References m33::e.

Referenced by m_rotmat().

5.5.1.16 void m_matadd (struct m33 * *m1*, struct m33 * *m2*)

3x3 Matrix addition m1+m2, result in m1

Definition at line 140 of file vm.c.

References m33::e.

5.5.1.17 void m_print (struct m33 * *m1*)

Print 3x3 matrix m1 to stdout

Definition at line 151 of file vm.c.

References m33::e.

5.6 Front-end interface

Files

- file **bpm_interface.h**

  Front end interface structure definitions and handlers.

- file **get_header.c**
- file **load_bpmconf.c**
- file **load_signals.c**
- file **load_struct.c**
- file **save_signals.c**

Data Structures

- struct **bpmconf**
- struct **bpmsignal**
- struct **bpmcalib**
- struct **bpmproc**
- struct **beamconf**

Typedefs

- typedef **bpmconf bpmconf_t**
- typedef **bpmsignal bpmsignal_t**
- typedef **bpmcalib bpmcalib_t**
- typedef **bpmproc bpmproc_t**
- typedef **beamconf beamconf_t**

Enumerations

- enum **bpmtype_t** { **diode**, **monopole**, **dipole** }
- enum **bpmpol_t** { **horiz**, **vert** }
- enum **bpmphase_t** { **randomised**, **locked** }
- enum **rffiltertype_t** { **nofilter**, **butterworth_low_pass**, **butterworth_band_pass**, **butterworth_high_pass** }

Functions

- EXTERN int **load_bpmconf** (const char *fname, **bpmconf_t** **conf, int *num_conf)
- EXTERN int **get_header** (FILE *file, double *version, int *num_structs)
- EXTERN int **load_struct** (FILE *file, char ***arg_list, char ***val_list, int *num_args)
- EXTERN int **save_signals** (char *fname, **bpmsignal_t** *sigs, int num_evts)
- EXTERN int **load_signals** (char *fname, **bpmsignal_t** **sigs)

Variables

- EXTERN int **bpm_verbose**

5.6.1 Typedef Documentation

5.6.1.1 typedef struct bpmconf bpmconf_t

type definition for BPM configuration

Definition at line 75 of file bpm_interface.h.

5.6.1.2 typedef struct bpmsignal bpmsignal_t

type definition for BPM signals

Definition at line 76 of file bpm_interface.h.

5.6.1.3 typedef struct bpmcalib bpmcalib_t

type definition for calibrations

Definition at line 77 of file bpm_interface.h.

5.6.1.4 typedef struct bpmproc bpmproc_t

type definition for processed BPM signals

Definition at line 78 of file bpm_interface.h.

5.6.1.5 typedef struct beamconf beamconf_t

type definition for beam configurations

Definition at line 79 of file bpm_interface.h.

5.6.2 Enumeration Type Documentation

5.6.2.1 enum bpmtype_t

BPM cavity (of better signal) type

Enumerator:

diode diodified bpm signal or trigger pulse

monopole reference cavity signal (monopole)

dipole position sentivive cavity signal (dipole)

Definition at line 40 of file bpm_interface.h.

5.6.2.2 enum bpmpol_t

BPM polarisation plane, basically a difficult way to say x or y ;)

Enumerator:

horiz Horizontal plane, or x in most cases

vert Vertical plane, or y in most cases

Definition at line 49 of file bpm_interface.h.

5.6.2.3 enum bpmphase_t

BPM electronics phase lock type

Enumerator:

randomised unlocked phase

locked locked phase

Definition at line 57 of file bpm_interface.h.

5.6.2.4 enum rffiltertype_t

RF filter type

PROBABLY MADE REDUNDANT BY BPMDSP, NEED TO CHECK THIS !!

Enumerator:

nofilter Don't filter the signal

butterworth_low_pass Butterworth low pass - pars: order, freq cutoff

butterworth_band_pass Butterworth band pass - pars: order, central freq, bandwidth

butterworth_high_pass Butterworth high pass - pars: order, freq cutoff

Definition at line 67 of file bpm_interface.h.

5.6.3 Function Documentation

5.6.3.1 EXTERN int load_bpmconf (const char * *fname*, bpmconf_t ** *conf*, int * *num_conf*)

Load a set of bpm configurations from file fname. Memory is allocated using calloc and so is the responsibility of the user to delete after use.

The configuration file lists the fields and their initial values. The first non-comment line is the header for the configuration. Hashed lines indicate comments.

Example of a bpmconf file:

Header - libbpm version, number of BPMs 0.1 21

Here are the BPM definitions themselves. Add whichever you want though the # ** fields are required. # Everything else will be set to -DBL_MAX bpm_x9 # BPM name. Currently not used. { bpm_idx 0 # The index in the created array **

cav_type dipole cav_polarisation horiz cav_phasetype locked cav_freq 2626 # Cavity frequency (in MHz) cav_decaytime 3 # Decay time (microsec) cav_phase 0 cav_iqrotation 0 cav_chargesens 10 cav_possens 10 cav_tiltsens 10

rf_LOfreq 2550 rf_filtertype 0 rf_nfiltpars 4 rf_filterpars 10 10 10 10

digi_trigtimeoffset 50 digi_freq 100 digi_nbits 14 # Number of bits in the ADC ** digi_nsamples 256 # Number of samples in the ADC ** digi_ampnoise 5 digi_voltageoffset 8192 digi_phasenoise 3

geom_pos 0 0 40 geom_tilt 0 0 0

ref_idx 10 # Refernce index ** diode_idx 10 # Diode index ** }

etc...

Parameters:

fname the filename of the configuration file to load

conf the pointer to the newly created set of configurations

num_conf the number of conifgurations loaded

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 12 of file load_bpmconf.c.

References bpm_error(), bpm_warning(), butterworth_band_pass, butterworth_high_pass, butterworth_low_pass, diode, dipole, get_header(), horiz, load_struct(), locked, MHz, monopole, nofilter, randomised, and vert.

5.6.3.2 EXTERN int get_header (FILE * *file*, double * *version*, int * *num_structs*)

Load in the header information from a configuration file. The header must have the bpm version followed by the number of entries. Comments are denoted by #

Example of the header:

Header - libbpm version, number of BPMs 0.1 21

Parameters:

file A FILE pointer to the stream to load from

version Pointer to a double that is filled with the version number

num_structs Pointer to an integer that is filled with the number of structs in the file

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 9 of file get_header.c.

References bpm_error().

Referenced by load_bpmconf(), and load_signals().

5.6.3.3 EXTERN int load_struct (FILE * *file*, char *** *arg_list*, char *** *val_list*, int * *num_args*)

Load in a structure from a file and return the arguments and the values in a list. Comments are denoted by #

Example of a structure:

Describe x9 using a bpmconf struture x9

Parameters:

file A FILE pointer to the stream to load from

arg_list Pointer to an array of names that will hold the arguments

val_list Pointer to an array of the values for each field specfied in arg_list

num_args Pointer to an integer that will hold the number of arguments found

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 11 of file load_struct.c.

References bpm_error(), and MAX_ARGS.

Referenced by load_bpmconf().

5.6.3.4 EXTERN int save_signals (char * *fname*, bpmsignal_t * *sigs*, int *num_evts*)

Save a set of waveforms

Parameters:

fname The filename to save to

sigs The bpmsignal structures to save

num_evts The number of events

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 10 of file save_signals.c.

References bpm_error(), and bpmsignal::ns.

5.6.3.5 EXTERN int load_signals (char * *fname*, bpmsignal_t ** *sigs*)

Load the specified number of events from the given file

Parameters:

fname The filename to load from

sigs The bpmsignal structures

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 10 of file load_signals.c.

References bpm_error(), bpm_warning(), and get_header().

5.6.4 Variable Documentation

5.6.4.1 EXTERN int bpm_verbose

be a bit verbose in libbpm

Definition at line 206 of file bpm_interface.h.

Referenced by get_t0().

5.7 Error/warning messages

Files

- file **bpm_error.c**
- file **bpm_messages.h**
  libbpm error/warning messages
- file **bpm_warning.c**

Functions

- EXTERN void **bpm_error** (char *msg, char *f, int l)
- EXTERN void **bpm_warning** (char *msg, char *f, int l)

5.7.1 Function Documentation

5.7.1.1 EXTERN void bpm_error (char * *msg*, char * *f*, int *l*)

Prints an error message in a standard format

Parameters:

msg the error messages, without end of line character

f the file position (__FILE__)

l the line in the file (__LINE__)

Returns:

void

Definition at line 8 of file bpm_error.c.

Referenced by _expand_complex_polynomial(), add_amplnoise(), add_excitation(), add_scalar_waveform(), add_wave(), add_waveforms(), alloc_complex_wave_double(), alloc_simple_wave_double(), alloc_simple_wave_int(), ana_set_cutfn(), apply_filter(), basic_stats(), calibrate(), calibrate_simple(), calibrate_svd(), cblas_dgemv(), copy_waveform(), create_filter(), create_resonator_representation(), create_splane_representation(), ddc_gaussfilter(), ddc_gaussfilter_step(), ddc_sample_waveform(), ddc_waveform(), digitise(), downmix_waveform(), fft_waveform_double(), filter_impulse_response(), filter_step_response(), fit_ddc(), fit_fft(), fit_fft_prepare(), fit_waveform(), generate_bpm_orbit(), generate_corr_scan(), generate_diode(), generate_dipole(), generate_monopole(), generate_mover_scan(), generate_noise(), get_amplitude(), get_complex_from_AmpPhi(), get_complex_from_ReIm(), get_header(), get_imaginary_part(), get_IQ(), get_pedestal(), get_phase(), get_pos(), get_real_part(), get_slope(), get_t0(), gsl_blas_dgemv(), gsl_block_alloc(), gsl_linalg_bidiag_decomp(), gsl_linalg_bidiag_unpack(), gsl_linalg_bidiag_unpack2(), gsl_linalg_SV_decomp(), gsl_linalg_SV_solve(), gsl_matrix_alloc(), gsl_matrix_column(), gsl_matrix_const_column(), gsl_matrix_const_row(), gsl_matrix_row(), gsl_matrix_submatrix(), gsl_matrix_swap_columns(), gsl_vector_alloc(), gsl_vector_const_subvector(), gsl_vector_subvector(), gsl_vector_swap_elements(), handle_saturation(), int_to_double_waveform(), load_bpmconf(), load_calibration(), load_signals(), load_struct(), mult_scalar_waveform(), mult_waveform(), normalise_filter(), nr_ax_eq_b_LU(), nr_fit(), nr_four1(), nr_gammln(), nr_gammq(), nr_gcf(), nr_gser(), nr_lmchkjac(), nr_lmder(), nr_lmder_bc(), nr_lmdif(), nr_lmdif_bc(), nr_median(), nr_realft(), nr_seed(),

nr_select(), print_filter(), process_diode(), process_dipole(), process_waveform(), reset_complex_wave(), reset_simple_wave(), rf_addLO(), rf_amplify(), rf_butterworthbandpass(), rf_butterworthhighpass(), rf_butterworthlowpass(), rf_complexFFT(), rf_filter(), rf_mixer(), rf_rectify(), save_calibration(), save_signals(), setup_calibration(), simple_tone(), simple_wave(), update_freq_tdecay(), and zplane_transform().

5.7.1.2 EXTERN void bpm_warning (char * *msg*, char * *f*, int *l*)

Prints an warning message in a standard format

Parameters:

msg the error messages, without end of line character

f the file position (__FILE__)

l the line in the file (__LINE__)

Returns:

void

Definition at line 8 of file bpm_warning.c.

Referenced by basic_stats(), create_filter(), ddc_gaussfilter_step(), generate_corr_scan(), generate_mover_scan(), get_IQ(), get_pedestal(), get_t0(), handle_saturation(), load_bpmconf(), load_signals(), nr_gcf(), nr_gser(), and process_waveform().

5.8 Numerical routines

Files

- file **bpm_nr.h**
  libbpm numerical helper routines
- file **gsl_blas.c**
- file **gsl_block.c**
- file **gsl_eigen.c**
- file **gsl_linalg.c**
- file **gsl_matrix.c**
- file **gsl_vector.c**
- file **nr_checks.c**
- file **nr_complex.c**
- file **nr_fit.c**
- file **nr_four1.c**
- file **nr_gammln.c**
- file **nr_gammq.c**
- file **nr_gcf.c**
- file **nr_gser.c**
- file **nr_levmar.c**
- file **nr_median.c**
- file **nr_ran1.c**
- file **nr_rangauss.c**
- file **nr_ranuniform.c**

- file **nr_realft.c**
- file **nr_seed.c**
- file **nr_select.c**

Data Structures

- struct **lm_fstate**
- struct **gsl_block_struct**
- struct **gsl_matrix**
- struct **_gsl_matrix_view**
- struct **gsl_vector**
- struct **_gsl_vector_view**
- struct **_gsl_vector_const_view**
- struct **complex_t**

Defines

- #define **GCF_ITMAX**
- #define **GCF_FPMIN**
- #define **GCF_EPS**
- #define **GSER_EPS**
- #define **GSER_ITMAX**
- #define **RAN1_IA**
- #define **RAN1_IM**
- #define **RAN1_AM**
- #define **RAN1_IQ**
- #define **RAN1_IR**
- #define **RAN1_NTAB**
- #define **RAN1_NDIV**
- #define **RAN1_EPS**
- #define **RAN1_RNMX**
- #define **__LM_BLOCKSZ__**
- #define **__LM_BLOCKSZ__SQ**
- #define **LINSOLVERS_RETAIN_MEMORY**
- #define **__LM_STATIC__**
- #define **FABS**(x)
- #define **CNST**(x)
- #define **_LM_POW_**
- #define **LM_DER_WORKSZ**(npar, nmeas)
- #define **LM_DIF_WORKSZ**(npar, nmeas)
- #define **LM_EPSILON**
- #define **LM_ONE_THIRD**
- #define **LM_OPTS_SZ**
- #define **LM_INFO_SZ**
- #define **LM_INIT_MU**
- #define **LM_STOP_THRESH**
- #define **LM_DIFF_DELTA**
- #define **NR_FFTFORWARD**
- #define **NR_FFTBACKWARD**
- #define **__LM_MEDIAN3**(a, b, c)

- #define **NULL_VECTOR**
- #define **NULL_VECTOR_VIEW**
- #define **NULL_MATRIX**
- #define **NULL_MATRIX_VIEW**
- #define **GSL_DBL_EPSILON**
- #define **OFFSET**(N, incX)
- #define **GSL_MIN**(a, b)

Typedefs

- typedef enum **CBLAS_TRANSPOSE CBLAS_TRANSPOSE_t**
- typedef **gsl_block_struct gsl_block**
- typedef **_gsl_matrix_view gsl_matrix_view**
- typedef **_gsl_vector_view gsl_vector_view**
- typedef const **_gsl_vector_const_view gsl_vector_const_view**

Enumerations

- enum **CBLAS_TRANSPOSE** { **CblasNoTrans**, **CblasTrans**, **CblasConjTrans** }
- enum **CBLAS_ORDER** { **CblasRowMajor**, **CblasColMajor** }

Functions

- EXTERN double **nr_gammln** (double xx)
- EXTERN double **nr_gammq** (double a, double x)
- EXTERN int **nr_gcf** (double *gammcf, double a, double x, double *gln)
- EXTERN int **nr_gser** (double *gamser, double a, double x, double *gln)
- EXTERN int **nr_fit** (double *x, double y[], int ndata, double sig[], int mwt, double *a, double *b, double *siga, double *sigb, double *chi2, double *q)
- EXTERN int **nr_is_pow2** (unsigned long n)
- EXTERN int **nr_four1** (double data[], unsigned long nn, int isign)
- EXTERN int **nr_realft** (double data[], unsigned long n, int isign)
- EXTERN double **nr_ran1** (long *idum)
- EXTERN int **nr_seed** (long seed)
- EXTERN double **nr_ranuniform** (double lower, double upper)
- EXTERN double **nr_rangauss** (double mean, double std_dev)
- EXTERN int **nr_lmder** (void(*func)(double *p, double *hx, int m, int n, void *adata), void(*jacf)(double *p, double *j, int m, int n, void *adata), double *p, double *x, int m, int n, int itmax, double *opts, double *info, double *work, double *covar, void *adata)
- EXTERN int **nr_lmdif** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *x, int m, int n, int itmax, double *opts, double *info, double *work, double *covar, void *adata)
- EXTERN int **nr_lmder_bc** (void(*func)(double *p, double *hx, int m, int n, void *adata), void(*jacf)(double *p, double *j, int m, int n, void *adata), double *p, double *x, int m, int n, double *lb, double *ub, int itmax, double *opts, double *info, double *work, double *covar, void *adata)
- EXTERN int **nr_lmdif_bc** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *x, int m, int n, double *lb, double *ub, int itmax, double *opts, double *info, double *work, double *covar, void *adata)

- EXTERN void **nr_lmchkjac** (void(*func)(double *p, double *hx, int m, int n, void *adata), void(*jacf)(double *p, double *j, int m, int n, void *adata), double *p, int m, int n, void *adata, double *err)
- EXTERN int **nr_lmcovar** (double *JtJ, double *C, double sumsq, int m, int n)
- EXTERN int **nr_ax_eq_b_LU** (double *A, double *B, double *x, int n)
- EXTERN void **nr_trans_mat_mat_mult** (double *a, double *b, int n, int m)
- EXTERN void **nr_fdif_forw_jac_approx** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *hx, double *hxx, double delta, double *jac, int m, int n, void *adata)
- EXTERN void **nr_fdif_cent_jac_approx** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *hxm, double *hxp, double delta, double *jac, int m, int n, void *adata)
- EXTERN double **nr_median** (int n, double *arr)
- EXTERN double **nr_select** (int k, int n, double *org_arr)
- EXTERN **gsl_matrix** * **gsl_matrix_calloc** (const size_t n1, const size_t n2)
- EXTERN **_gsl_vector_view gsl_matrix_column** (**gsl_matrix** *m, const size_t i)
- EXTERN **_gsl_matrix_view gsl_matrix_submatrix** (**gsl_matrix** *m, const size_t i, const size_t j, const size_t n1, const size_t n2)
- EXTERN double **gsl_matrix_get** (const **gsl_matrix** *m, const size_t i, const size_t j)
- EXTERN void **gsl_matrix_set** (**gsl_matrix** *m, const size_t i, const size_t j, const double x)
- EXTERN int **gsl_matrix_swap_columns** (**gsl_matrix** *m, const size_t i, const size_t j)
- EXTERN **gsl_matrix** * **gsl_matrix_alloc** (const size_t n1, const size_t n2)
- EXTERN **_gsl_vector_const_view gsl_matrix_const_row** (const **gsl_matrix** *m, const size_t i)
- EXTERN **_gsl_vector_view gsl_matrix_row** (**gsl_matrix** *m, const size_t i)
- EXTERN **_gsl_vector_const_view gsl_matrix_const_column** (const **gsl_matrix** *m, const size_t j)
- EXTERN void **gsl_matrix_set_identity** (**gsl_matrix** *m)
- EXTERN **gsl_vector** * **gsl_vector_calloc** (const size_t n)
- EXTERN **_gsl_vector_view gsl_vector_subvector** (**gsl_vector** *v, size_t offset, size_t n)
- EXTERN double **gsl_vector_get** (const **gsl_vector** *v, const size_t i)
- EXTERN void **gsl_vector_set** (**gsl_vector** *v, const size_t i, double x)
- EXTERN int **gsl_vector_swap_elements** (**gsl_vector** *v, const size_t i, const size_t j)
- EXTERN **_gsl_vector_const_view gsl_vector_const_subvector** (const **gsl_vector** *v, size_t i, size_t n)
- EXTERN void **gsl_vector_free** (**gsl_vector** *v)
- EXTERN int **gsl_linalg_SV_solve** (const **gsl_matrix** *U, const **gsl_matrix** *Q, const **gsl_vector** *S, const **gsl_vector** *b, **gsl_vector** *x)
- EXTERN int **gsl_linalg_bidiag_unpack** (const **gsl_matrix** *A, const **gsl_vector** *tau_U, **gsl_matrix** *U, const **gsl_vector** *tau_V, **gsl_matrix** *V, **gsl_vector** *diag, **gsl_vector** *superdiag)
- EXTERN int **gsl_linalg_householder_hm** (double tau, const **gsl_vector** *v, **gsl_matrix** *A)
- EXTERN int **gsl_linalg_bidiag_unpack2** (**gsl_matrix** *A, **gsl_vector** *tau_U, **gsl_vector** *tau_V, **gsl_matrix** *V)
- EXTERN int **gsl_linalg_householder_hm1** (double tau, **gsl_matrix** *A)
- EXTERN void **create_givens** (const double a, const double b, double *c, double *s)

- EXTERN double **gsl_linalg_householder_transform** (**gsl_vector** *v)
- EXTERN int **gsl_linalg_householder_mh** (double tau, const **gsl_vector** *v, **gsl_matrix** *A)
- EXTERN void **chop_small_elements** (**gsl_vector** *d, **gsl_vector** *f)
- EXTERN void **qrstep** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, **gsl_matrix** *V)
- EXTERN double **trailing_eigenvalue** (const **gsl_vector** *d, const **gsl_vector** *f)
- EXTERN void **create_schur** (double d0, double f0, double d1, double *c, double *s)
- EXTERN void **svd2** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, **gsl_matrix** *V)
- EXTERN void **chase_out_intermediate_zero** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, size_t k0)
- EXTERN void **chase_out_trailing_zero** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *V)
- EXTERN int **gsl_isnan** (const double x)
- EXTERN double **gsl_blas_dnrm2** (const **gsl_vector** *X)
- EXTERN double **cblas_dnrm2** (const int N, const double *X, const int incX)
- EXTERN void **gsl_blas_dscal** (double alpha, **gsl_vector** *X)
- EXTERN void **cblas_dscal** (const int N, const double alpha, double *X, const int incX)
- EXTERN void **cblas_dgemv** (const enum **CBLAS_ORDER** order, const enum **CBLAS_TRANSPOSE** TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)
- EXTERN **gsl_block** * **gsl_block_alloc** (const size_t n)
- EXTERN void **gsl_block_free** (**gsl_block** *b)
- EXTERN **complex_t complex** (double re, double im)
- EXTERN double **c_real** (**complex_t** z)
- EXTERN double **c_imag** (**complex_t** z)
- EXTERN **complex_t c_conj** (**complex_t** z)
- EXTERN **complex_t c_neg** (**complex_t** z)
- EXTERN **complex_t c_sum** (**complex_t** z1, **complex_t** z2)
- EXTERN **complex_t c_diff** (**complex_t** z1, **complex_t** z2)
- EXTERN **complex_t c_mult** (**complex_t** z1, **complex_t** z2)
- EXTERN **complex_t c_div** (**complex_t** z1, **complex_t** z2)
- EXTERN **complex_t c_scale** (double r, **complex_t** z)
- EXTERN **complex_t c_sqr** (**complex_t** z)
- EXTERN **complex_t c_sqrt** (**complex_t** z)
- EXTERN double **c_norm2** (**complex_t** z)
- EXTERN double **c_abs** (**complex_t** z)
- EXTERN double **c_arg** (**complex_t** z)
- EXTERN **complex_t c_exp** (**complex_t** z)
- EXTERN int **c_isequal** (**complex_t** z1, **complex_t** z2)

Variables

- EXTERN long **bpm_rseed**

5.8.1 Define Documentation

5.8.1.1 #define __LM_BLOCKSZ__

Block size for cache-friendly matrix-matrix multiply. It should be such that __BLOCKSZ__^2*sizeof(LM_REAL) is smaller than the CPU (L1) data cache size. Notice that a value of 32 when LM_REAL=double assumes an 8Kb L1 data cache (32*32*8=8K). This is a concervative choice since newer Pentium 4s have a L1 data cache of size 16K, capable of holding up to 45x45 double blocks.

Definition at line 55 of file bpm_nr.h.

Referenced by nr_trans_mat_mat_mult().

5.8.1.2 #define LM_DER_WORKSZ(npar, nmeas)

Work array size for LM with & without jacobian, should be multiplied by sizeof(double) or sizeof(float) to be converted to bytes

Definition at line 73 of file bpm_nr.h.

Referenced by nr_lmder(), and nr_lmder_bc().

5.8.1.3 #define LM_DIF_WORKSZ(npar, nmeas)

see LM_DER_WORKSZ

Definition at line 75 of file bpm_nr.h.

Referenced by nr_lmdif().

5.8.1.4 #define NR_FFTFORWARD

Perform forward FFT in nr_four

Definition at line 86 of file bpm_nr.h.

Referenced by rf_butterworthhighpass(), and rf_butterworthlowpass().

5.8.1.5 #define NR_FFTBACKWARD

Perform backward FFT in nr_four

Definition at line 87 of file bpm_nr.h.

Referenced by rf_butterworthhighpass(), and rf_butterworthlowpass().

5.8.1.6 #define __LM_MEDIAN3(a, b, c)

find the median of 3 numbers

Definition at line 90 of file bpm_nr.h.

5.8.2 Function Documentation

5.8.2.1 EXTERN double nr_gammln (double *xx*)

Calculates the logaritm of the gamma function ln[gamma(xx)]. NR C6.1, p 214 supposed to be correct to double precision

Parameters:

xx the argument

Returns:

the value of ln[gamma(xx)]

Definition at line 16 of file nr_gammln.c.

References bpm_error(), and nr_is_int().

Referenced by nr_gcf(), and nr_gser().

5.8.2.2 EXTERN double nr_gammq (double *a*, double *x*)

Returns the incomplete gamma function. From numerical recipes, C6.2, p218

Returns:

-DBL_MAX upon failure

Definition at line 14 of file nr_gammq.c.

References bpm_error(), nr_gcf(), and nr_gser().

Referenced by nr_fit().

5.8.2.3 EXTERN int nr_gcf (double * *gammcf*, double *a*, double *x*, double * *gln*)

Returns the incomplete gamma function NR C6.2, p219

Definition at line 11 of file nr_gcf.c.

References bpm_error(), bpm_warning(), GCF_EPS, GCF_FPMIN, GCF_ITMAX, and nr_gammln().

Referenced by nr_gammq().

5.8.2.4 EXTERN int nr_gser (double * *gamser*, double *a*, double *x*, double * *gln*)

Returns incomplete gamma function. NR 6.2, 218

Definition at line 11 of file nr_gser.c.

References bpm_error(), bpm_warning(), GSER_EPS, GSER_ITMAX, and nr_gammln().

Referenced by nr_gammq().

5.8.2.5 EXTERN int nr_fit (double * *x*, double *y*[ ], int *ndata*, double *sig*[ ], int *mwt*, double * *a*, double * *b*, double * *siga*, double * *sigb*, double * *chi2*, double * *q*)

Fit data to a straight line. Nicked from numerical recipes, C15.2, p665 See: `http://www.library.cornell.edu/nr/cbookcpdf.html`

Parameters:

x array with x values

y array with corresponding y values

ndata number of datapoints

sig array with errors on y datapoints

mwt used weighted (so including errors on datapoints ?)

a fitted slope

b fitted intercept

siga error on fitted slope

sigb error on fitted intercept

chi2 chi2 of fit

q quality factor of fit

Returns:

BPM_FAILURE upon failure, BPM_SUCCESS upon success

Definition at line 27 of file nr_fit.c.

References bpm_error(), and nr_gammq().

Referenced by calibrate(), and get_t0().

5.8.2.6 EXTERN int nr_is_pow2 (unsigned long *n*)

Checks whether the input argument is an integer power of 2, like 256, 1024 etc...

Parameters:

n given unsigend long argument for which to check this

Returns:

FALSE if not a power of 2. The routine returns the precise power (> 1) if the integer is indeed a power of 2

Definition at line 39 of file nr_checks.c.

Referenced by nr_four1(), and nr_realft().

5.8.2.7 EXTERN int nr_four1 (double *data*[], unsigned long *nn*, int *isign*)

Replaces data[1..2∗nn] by its discrete Fourier transform, if isign is input as 1, or replaces data[1..2∗nn] by nn times its inverse discrete Fourier transform if isign is input as -1.

data is a complex arry of length nn, or equivalently a real array of length 2∗nn. nn MUST !!! be an integer power of 2, this is not checked for...

BM. 15.08.2005... added this check ;-))

Perform an FFT, NR S12.2 pg507 See: `http://www.library.cornell.edu/nr/cbookcpdf.html`

Parameters:

data array with data

nn number of data points, note that the array length has to be at least twice this number

isign sign of transform

Returns:

BPM_FAILURE upon failure, BPM_SUCCESS upon success

Definition at line 32 of file nr_four1.c.

References bpm_error(), and nr_is_pow2().

Referenced by fft_waveform_double(), nr_realft(), and rf_complexFFT().

5.8.2.8 EXTERN int nr_realft (double *data*[], unsigned long *n*, int *isign*)

Calculates the Fourier transform on a set of n real valued datapoints replaces this data (array data[1..n] by the positive frequency half of its complex Fourier transform. The real valued first and last components of the complex tranform are returned as elements data[1] and data[2] respectively, n MUST be a power of 2. This routines calculates the inverse transform of a complex data array if it is the transform of real data, result in this case must be multiplied with 2/n

BM. 15.08.2006: added the 2^n check on n Compute the FFT of a real function. NR 12.3 pg513

Parameters:

data the array with the data, which gets replaced by fft

n length of the data, must be power of 2

isign sign of the transform

Returns:

BPM_FAILURE upon failure, BPM_SUCCESS upon success

Definition at line 27 of file nr_realft.c.

References bpm_error(), nr_four1(), and nr_is_pow2().

5.8.2.9 EXTERN double nr_ran1 (long * *idum*)

Random number generator as nicked from numerical recipes, c7.1, p280

Parameters:

idum random seed, note that the global seed is set by bpm_rseed

Returns:

random number between 0 and 1

Definition at line 13 of file nr_ran1.c.

References RAN1_AM, RAN1_IA, RAN1_IM, RAN1_IQ, RAN1_IR, RAN1_NDIV, RAN1_NTAB, and RAN1_RNMX.

Referenced by nr_rangauss(), and nr_ranuniform().

5.8.2.10 EXTERN int nr_seed (long *seed*)

Set the random seed 'idum' to enable other random functions to work

Parameters:

seed a random seed

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 19 of file nr_seed.c.

References bpm_error(), and bpm_rseed.

5.8.2.11 EXTERN double nr_ranuniform (double *lower*, double *upper*)

Sample from a uniform distribution between (and exluding) the upper and lower values.

Parameters:

lower the lower range for the generation

upper the upper range for the generation

Returns:

the value of the uniform deviate, returns -DBL_MAX if the seed was not set correctly before

Definition at line 18 of file nr_ranuniform.c.

References bpm_rseed, and nr_ran1().

Referenced by add_amplnoise(), and rf_addLO().

5.8.2.12 EXTERN double nr_rangauss (double *mean*, double *std_dev*)

Sample a given Gaussian distribution using ran1 as the source of the uniform deviate between 0 and 1. Nicked from numerical recipes, C7.2, p289

Parameters:

mean the mean of the gaussian

std_dev the standard deviation of the gaussian

Returns:

a gaussian deviate, returns -DBL_MAX if the random seed is not set properly before

Definition at line 19 of file nr_rangauss.c.

References bpm_rseed, and nr_ran1().

Referenced by add_amplnoise(), digitise(), rf_addLO(), simple_tone(), and simple_wave().

5.8.2.13 EXTERN double nr_median (int *n*, double * *arr*)

Find the median value of the given array. Basically a wrapper for nr_select

Returns:

The value of the median element

Definition at line 13 of file nr_median.c.

References bpm_error(), and nr_select().

5.8.2.14 EXTERN double nr_select (int *k*, int *n*, double * *org_arr*)

Find the kth largest element of the array after sorting. Nicked from numerical recipes, C8.5, p342 See: `http://www.library.cornell.edu/nr/cbookcpdf.html`

Returns:

The value of the median element

Definition at line 14 of file nr_select.c.

References bpm_error().

Referenced by nr_median().

5.8.2.15 EXTERN _gsl_vector_view gsl_matrix_column (gsl_matrix * *m*, const size_t *j*)

Retrieve a column of a matrix

Parameters:

m The matrix

j index of the column

Returns:

BPM_SUCCESS if everything was OK, BPM_FAILURE if not

Definition at line 90 of file gsl_matrix.c.

References gsl_vector::block, gsl_matrix::block, bpm_error(), gsl_vector::data, gsl_matrix::data, NULL_VECTOR, NULL_VECTOR_VIEW, gsl_vector::owner, gsl_vector::size, gsl_matrix::size1, gsl_matrix::size2, gsl_vector::stride, gsl_matrix::tda, and _gsl_vector_view::vector.

Referenced by chase_out_intermediate_zero(), chase_out_trailing_zero(), gsl_linalg_bidiag_decomp(), gsl_linalg_householder_hm(), gsl_linalg_householder_hm1(), gsl_linalg_SV_decomp(), and qrstep().

5.8.2.16 EXTERN _gsl_matrix_view gsl_matrix_submatrix (gsl_matrix * *m*, const size_t *i*, const size_t *j*, const size_t *n1*, const size_t *n2*)

Retrieve a submatrix of the given matrix

Definition at line 152 of file gsl_matrix.c.

References gsl_matrix::block, bpm_error(), gsl_matrix::data, _gsl_matrix_view::matrix, NULL_MATRIX, NULL_MATRIX_VIEW, gsl_matrix::owner, gsl_matrix::size1, gsl_matrix::size2, and gsl_matrix::tda.

Referenced by gsl_linalg_bidiag_decomp(), gsl_linalg_bidiag_unpack(), gsl_linalg_bidiag_unpack2(), gsl_linalg_householder_hm(), gsl_linalg_householder_hm1(), gsl_linalg_householder_mh(), and gsl_linalg_SV_decomp().

5.8.2.17 EXTERN double gsl_matrix_get (const gsl_matrix * *m*, const size_t *i*, const size_t *j*)

Get the matrix value associated with the given row and column

Parameters:

m The matrix

i The row number

j The column number

Returns:

The value of the matrix element

Definition at line 124 of file gsl_matrix.c.

References gsl_matrix::data, and gsl_matrix::tda.

Referenced by chase_out_intermediate_zero(), chase_out_trailing_zero(), gsl_linalg_bidiag_unpack(), gsl_linalg_bidiag_unpack2(), gsl_linalg_householder_hm(), gsl_linalg_householder_hm1(), gsl_linalg_householder_mh(), gsl_linalg_SV_decomp(), qrstep(), and svd2().

5.8.2.18 EXTERN void gsl_matrix_set (gsl_matrix * *m*, const size_t *i*, const size_t *j*, const double *x*)

Set the matrix value associated with the given row and column

Parameters:

m The matrix

i The row number

j The column number

x the value to set

Definition at line 141 of file gsl_matrix.c.

References gsl_matrix::data, and gsl_matrix::tda.

Referenced by ana_get_svd_coeffs(), chase_out_intermediate_zero(), chase_out_trailing_zero(), gsl_linalg_householder_hm(), gsl_linalg_householder_hm1(), gsl_linalg_householder_mh(), gsl_linalg_SV_decomp(), qrstep(), and svd2().

5.8.2.19 EXTERN int gsl_matrix_swap_columns (gsl_matrix * *m*, const size_t *i*, const size_t *j*)

Swap two matrix columns

Parameters:

m The matrix

i index of column one

j index of column two

Returns:

BPM_SUCCESS if everything was OK, BPM_FAILURE if not

Definition at line 35 of file gsl_matrix.c.

References bpm_error(), gsl_matrix::data, gsl_matrix::size1, gsl_matrix::size2, and gsl_matrix::tda.

Referenced by gsl_linalg_SV_decomp(), and svd2().

5.8.2.20 EXTERN _gsl_vector_view gsl_vector_subvector (gsl_vector * *v*, size_t *offset*, size_t *n*)

Definition at line 8 of file gsl_vector.c.

References gsl_vector::block, bpm_error(), gsl_vector::data, NULL_VECTOR, NULL_VECTOR_VIEW, gsl_vector::owner, gsl_vector::size, gsl_vector::stride, and _gsl_vector_view::vector.

Referenced by gsl_linalg_bidiag_decomp(), gsl_linalg_householder_transform(), and gsl_linalg_SV_decomp().

5.8.2.21 EXTERN double gsl_vector_get (const gsl_vector * *v*, const size_t *i*)

Definition at line 61 of file gsl_vector.c.

References gsl_vector::data, and gsl_vector::stride.

Referenced by ana_get_svd_coeffs(), chase_out_intermediate_zero(), chase_out_trailing_zero(), chop_small_elements(), gsl_linalg_bidiag_unpack(), gsl_linalg_bidiag_unpack2(), gsl_linalg_householder_hm(), gsl_linalg_householder_mh(), gsl_linalg_householder_transform(), gsl_linalg_SV_decomp(), gsl_linalg_SV_solve(), qrstep(), svd2(), and trailing_eigenvalue().

5.8.2.22 EXTERN void gsl_vector_set (gsl_vector * *v*, const size_t *i*, double *x*)

Definition at line 70 of file gsl_vector.c.

References gsl_vector::data, and gsl_vector::stride.

Referenced by ana_get_svd_coeffs(), chase_out_intermediate_zero(), chase_out_trailing_zero(), chop_small_elements(), gsl_linalg_bidiag_decomp(), gsl_linalg_bidiag_unpack(), gsl_linalg_bidiag_unpack2(), gsl_linalg_householder_transform(), gsl_linalg_SV_decomp(), gsl_linalg_SV_solve(), qrstep(), and svd2().

5.8.2.23 EXTERN int gsl_linalg_householder_hm (double *tau*, const gsl_vector * *v*, gsl_matrix * *A*)

Definition at line 8 of file gsl_linalg.c.

References gsl_matrix_column(), gsl_matrix_get(), gsl_matrix_set(), gsl_matrix_submatrix(), gsl_vector_const_subvector(), gsl_vector_get(), _gsl_matrix_view::matrix,

gsl_vector::size, gsl_matrix::size1, gsl_matrix::size2, _gsl_vector_view::vector, and _gsl_vector_const_view::vector.

Referenced by gsl_linalg_bidiag_decomp(), gsl_linalg_bidiag_unpack(), and gsl_linalg_bidiag_unpack2().

5.8.2.24 EXTERN int gsl_linalg_householder_hm1 (double *tau*, gsl_matrix * *A*)

Definition at line 96 of file gsl_linalg.c.

References gsl_blas_dscal(), gsl_matrix_column(), gsl_matrix_get(), gsl_matrix_set(), gsl_matrix_submatrix(), _gsl_matrix_view::matrix, gsl_matrix::size1, gsl_matrix::size2, and _gsl_vector_view::vector.

Referenced by gsl_linalg_bidiag_unpack2().

5.8.2.25 EXTERN double gsl_linalg_householder_transform (gsl_vector * *v*)

Definition at line 285 of file gsl_linalg.c.

References gsl_blas_dnrm2(), gsl_blas_dscal(), gsl_vector_get(), gsl_vector_set(), gsl_vector_subvector(), gsl_vector::size, and _gsl_vector_view::vector.

Referenced by gsl_linalg_bidiag_decomp().

5.8.2.26 EXTERN int gsl_linalg_householder_mh (double *tau*, const gsl_vector * *v*, gsl_matrix * *A*)

Definition at line 322 of file gsl_linalg.c.

References gsl_matrix_get(), gsl_matrix_row(), gsl_matrix_set(), gsl_matrix_submatrix(), gsl_vector_const_subvector(), gsl_vector_get(), _gsl_matrix_view::matrix, gsl_vector::size, gsl_matrix::size1, gsl_matrix::size2, _gsl_vector_view::vector, and _gsl_vector_const_view::vector.

Referenced by gsl_linalg_bidiag_decomp().

5.8.2.27 EXTERN double gsl_blas_dnrm2 (const gsl_vector * *X*)

Definition at line 8 of file gsl_blas.c.

References cblas_dnrm2(), gsl_vector::data, gsl_vector::size, and gsl_vector::stride.

Referenced by gsl_linalg_householder_transform(), and gsl_linalg_SV_decomp().

5.8.2.28 EXTERN gsl_block* gsl_block_alloc (const size_t *n*)

Definition at line 8 of file gsl_block.c.

References bpm_error(), gsl_block_struct::data, and gsl_block_struct::size.

Referenced by gsl_matrix_alloc(), and gsl_vector_alloc().

5.9 BPM signal processing

Files

- file **add_scalar_waveform.c**

- file **basic_stats.c**
- file **bpm_process.h**

  libbpm main processing routines

- file **copy_waveform.c**
- file **ddc_gaussfilter.c**
- file **ddc_gaussfilter_step.c**
- file **ddc_sample_waveform.c**
- file **ddc_waveform.c**
- file **downmix_waveform.c**
- file **fft_waveform.c**
- file **fit_ddc.c**
- file **fit_diodepulse.c**
- file **fit_fft.c**
- file **fit_waveform.c**
- file **freq_to_sample.c**
- file **get_IQ.c**
- file **get_pedestal.c**
- file **get_pos.c**
- file **get_slope.c**
- file **get_t0.c**
- file **handle_saturation.c**
- file **int_to_double_waveform.c**
- file **mult_scalar_waveform.c**
- file **mult_waveform.c**
- file **process_diode.c**
- file **process_dipole.c**
- file **process_monopole.c**
- file **process_waveform.c**
- file **sample_to_freq.c**
- file **sample_to_time.c**
- file **time_to_sample.c**

Defines

- #define **PROC_DEFAULT**
- #define **PROC_DO_FFT**
- #define **PROC_DO_FIT**
- #define **PROC_DO_DDC**
- #define **PROC_DDC_CALIBFREQ**
- #define **PROC_DDC_CALIBTDECAY**
- #define **PROC_DDC_FITFREQ**
- #define **PROC_DDC_FITTDECAY**
- #define **PROC_DDC_FFTFREQ**
- #define **PROC_DDC_FFTTDECAY**
- #define **PROC_DDC_STOREFULL**
- #define **PROC_FIT_DDC**

Functions

- EXTERN int **process_diode** (**bpmconf_t** *, **bpmsignal_t** *, **bpmproc_t** *)
- EXTERN int **process_waveform** (enum **bpmtype_t** type, **bpmconf_t** *bpm, **bpmcalib_t** *cal, **bpmsignal_t** *sig, **bpmproc_t** *proc, **bpmproc_t** *trig, unsigned int mode)
- EXTERN int **process_monopole** (**bpmconf_t** *bpm, **bpmcalib_t** *cal, **bpmsignal_t** *sig, **bpmproc_t** *proc, **bpmproc_t** *trig, unsigned int mode)
- EXTERN int **process_dipole** (**bpmconf_t** *bpm, **bpmcalib_t** *cal, **bpmsignal_t** *sig, **bpmproc_t** *proc, **bpmproc_t** *trig, **bpmproc_t** *ref, unsigned int mode)
- EXTERN int **fit_waveform** (int *wf, int ns, double t0, double fs, double i_freq, double i_tdecay, double i_amp, double i_phase, double *freq, double *tdecay, double *amp, double *phase)
- EXTERN int **fit_diodepulse** (int *wf, int ns, double fs, double *t0)
- EXTERN int **fit_ddc** (double *ddc, int ns, double *tdecay)
- EXTERN int **fit_fft_prepare** (double **fft, int ns, double fs, int *n1, int *n2, double *amp, double *freq, double *fwhm)
- EXTERN int **fit_fft** (double **fft, int ns, double fs, double *freq, double *tdecay, double *A, double *C)
- EXTERN int **fft_waveform** (int *wf, int ns, double **fft)
- EXTERN int **fft_waveform_double** (double *wf, int ns, double **fft)
- EXTERN int **handle_saturation** (int *wf, int ns, int imax, int nbits, int threshold, int *iunsat)
- EXTERN int **downmix_waveform** (double *wf, int ns, double fs, double freq, double t0, double **out)
- EXTERN int **ddc_gaussfilter_step** (double **ddc, int ns, double fs, int istart, int istop, double tfilter, double filtBW, double *out)
- EXTERN int **ddc_gaussfilter** (double **ddc, int ns, double fs, double filtBW, double epsFilt, double **out)
- EXTERN int **ddc_waveform** (int *wf, int ns, int nbits, double fs, double t0, double freq, double tdecay, double filtBW, double epsFilt, double **out)
- EXTERN int **ddc_sample_waveform** (int *wf, int ns, int nbits, double fs, double t0, double t0Offset, double freq, double tdecay, double filtBW, double epsFilt, double *amp, double *phase)
- EXTERN int **get_pedestal** (int *wf, int ns, int range, double *offset, double *rms)
- EXTERN int **basic_stats** (int *wf, int ns, int range, int nbits, double *offset, double *rms, int *max, int *min, int *unsat_sample)
- EXTERN int **int_to_double_waveform** (double *wf_double, int *wf_int, int ns)
- EXTERN int **copy_waveform** (double *wf_src, double *wf_dst, int ns)
- EXTERN int **add_scalar_waveform** (double *wf, int ns, double add)
- EXTERN int **mult_scalar_waveform** (double *wf, int ns, double mult)
- EXTERN int **mult_waveform** (double *wf1, double *wf2, int ns)
- EXTERN int **get_t0** (int *wf, int ns, double fs, double *t0)
- EXTERN int **get_IQ** (double amp, double phase, double refamp, double refphase, double *Q, double *I)
- EXTERN int **get_pos** (double Q, double I, double IQphase, double posscale, double *pos)
- EXTERN int **get_slope** (double Q, double I, double IQphase, double slopescale, double *slope)
- EXTERN int **time_to_sample** (double fs, int ns, double t, int *iS)
- EXTERN int **sample_to_time** (double fs, int ns, int iS, double *t)
- EXTERN int **freq_to_sample** (double fs, int ns, double f, int *iS)
- EXTERN int **sample_to_freq** (double fs, int ns, int iS, double *f)

5.9.1 Function Documentation

5.9.1.1 EXTERN int process_diode (bpmconf_t * *bpm*, bpmsignal_t * *sig*, bpmproc_t * *proc*)

This routine processes a diode pulse, which should be found in the signal structure. It fills the proc structure with the t0.

Parameters:

bpm The bpm configuration structure

sig The bpm signal

proc The processed waveform structure

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 19 of file process_diode.c.

References bpm_error(), bpmconf::cav_type, bpmconf::digi_freq, bpmconf::digi_nsamples, diode, fit_diodepulse(), bpmconf::name, bpmproc::t0, and bpmsignal::wf.

5.9.1.2 EXTERN int process_waveform (enum bpmtype_t *type*, bpmconf_t * *bpm*, bpmcalib_t * *cal*, bpmsignal_t * *sig*, bpmproc_t * *proc*, bpmproc_t * *trig*, unsigned int *mode*)

Processes a general decaying sin wave according to the bitpattern given in mode the type needs to be specified to see whether the waveform type that is processed is correct to which what is expected. This routines is both used by process_monopole (which essentially does nothing more than wrap around this routine) and process_dipole which after this routine goes on to calculated the IQ and positions and tilt

Parameters:

type the bpm type

bpm the bpm configuration structure

cal the current valid calibration for the bpm

sig the waveform structure

proc the processed data structure

trig a pointer to the structure with processed trigger info for that waveform

mode processing mode

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 27 of file process_waveform.c.

References bpmproc::ampnoise, bpm_error(), bpm_warning(), bpmconf::cav_decaytime, bpmconf::cav_freq, bpmconf::cav_type, bpmproc::ddc_amp, bpmproc::ddc_phase, ddc_sample_waveform(), bpmproc::ddc_success, ddc_waveform(), bpmcalib::ddcepsFilt, bpmcalib::ddcfiltBW, bpmproc::ddcwf, bpmconf::digi_freq, bpmconf::digi_nbits, bpmconf::digi_nsamples, bpmproc::fft_freq, bpmproc::fft_success, bpmproc::fft_tdecay, fft_waveform(),

bpmproc::fftwf, bpmproc::fit_amp, fit_fft(), bpmproc::fit_freq, bpmproc::fit_phase, bpmproc::fit_success, bpmproc::fit_tdecay, fit_waveform(), bpmcalib::freq, get_pedestal(), handle_saturation(), MHz, bpmconf::name, nsec, PROC_DDC_FFTFREQ, PROC_DDC_FFTTDECAY, PROC_DDC_FITFREQ, PROC_DDC_FITTDECAY, PROC_DDC_STOREFULL, PROC_DO_DDC, PROC_DO_FFT, PROC_DO_FIT, bpmconf::rf_LOfreq, sample_to_time(), bpmproc::t0, bpmcalib::t0Offset, bpmcalib::tdecay, usec, bpmproc::voltageoffset, and bpmsignal::wf.

Referenced by process_dipole(), and process_monopole().

5.9.1.3 EXTERN int process_monopole (bpmconf_t * *bpm*, bpmcalib_t * *cal*, bpmsignal_t * *sig*, bpmproc_t * *proc*, bpmproc_t * *trig*, unsigned int *mode*)

Processes a monopole waveform according to the bitpattern given in mode. Is basically a wrapper for **process_waveform()** (p. 53) !

Parameters:

bpm the bpm configuration structure

cal the current valid calibration for the bpm

sig the waveform structure

proc the processed data structure

trig a pointer to the structure with processed trigger info for that waveform

mode a bitpattern encoding what exactly to process

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 23 of file process_monopole.c.

References monopole, and process_waveform().

5.9.1.4 EXTERN int process_dipole (bpmconf_t * *bpm*, bpmcalib_t * *cal*, bpmsignal_t * *sig*, bpmproc_t * *proc*, bpmproc_t * *trig*, bpmproc_t * *ref*, unsigned int *mode*)

Process dipole waveform

Parameters:

bpm Configuration structure for the bpm waveform to be processed

cal Calibration strcture with calib info to use

sig The BPM signal itself

proc The resuling processed signal

trig The already processed trigger waveform

ref The already processed reference waveform

mode Processing mode

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 22 of file process_dipole.c.

References bpm_error(), bpmproc::ddc_amp, bpmproc::ddc_I, bpmproc::ddc_phase, bpmproc::ddc_pos, bpmproc::ddc_Q, bpmproc::ddc_slope, bpmproc::ddc_success, dipole, bpmproc::fit_amp, bpmproc::fit_I, bpmproc::fit_phase, bpmproc::fit_pos, bpmproc::fit_Q, bpmproc::fit_slope, bpmproc::fit_success, get_IQ(), get_pos(), get_slope(), bpmcalib::IQphase, bpmconf::name, bpmcalib::posscale, process_waveform(), and bpmcalib::slopescale.

5.9.1.5 EXTERN int fit_waveform (int * *wf*, int *ns*, double *t0*, double *fs*, double *i_freq*, double *i_tdecay*, double *i_amp*, double *i_phase*, double * *freq*, double * *tdecay*, double * *amp*, double * *phase*)

Fits the waveform with a decaying sin wave using the lmder/lmdif routines from **nr_levmar.c** (p. 125) !

Parameters:

**wf* the waveform

ns number of samples

t0 t0 for the waveform

fs the sampling frequency

i_freq initial frequency for fit

i_tdecay initial tdecay

i_amp initial amp

i_phase initial phase

freq fitted frequency

tdecay fitted tdecay

amp fitted amplitude

phase fitted phase

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 101 of file fit_waveform.c.

References alloc_simple_wave_double(), bpm_error(), fcnwf(), fcnwfjac(), FIT_AMP, FIT_FREQ, FIT_FS, FIT_MAX_ITER, FIT_PHASE, FIT_T0, FIT_TDECAY, free_simple_wave_double(), get_pedestal(), LM_INFO_SZ, and LM_INIT_MU.

Referenced by process_waveform().

5.9.1.6 EXTERN int fit_diodepulse (int * *wf*, int *ns*, double *fs*, double * *t0*)

Fits the diode pulse, basically a wrapper for get_t0, to conserve names and consistency in the library...

see **get_t0()** (p. 61)

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 17 of file fit_diodepulse.c.

References get_t0().

Referenced by process_diode().

5.9.1.7 EXTERN int fit_ddc (double * *ddc*, int *ns*, double * *tdecay*)

Fits the ddc to get the decay time, gets inital pars from ddc wf itself

NOT IMPLEMENTED YET !

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 15 of file fit_ddc.c.

References bpm_error().

5.9.1.8 EXTERN int fit_fft_prepare (double ** *fft*, int *ns*, double *fs*, int * *n1*, int * *n2*, double * *amp*, double * *freq*, double * *fwhm*)

Prepares the fft fit of the waveform, fits only in the first nyquist band, scans the fft for the maximum value and returns !

Definition at line 77 of file fit_fft.c.

References bpm_error(), FIT_WINDOW_FACTOR, and MHz.

Referenced by fit_fft().

5.9.1.9 EXTERN int handle_saturation (int * *wf*, int *ns*, int *imax*, int *nbits*, int *threshold*, int * *iunsat*)

Handles the saturation, so computes the first sample where no saturation occurs, or imax if bigger...

Parameters:

wf the waveform

ns number of samples

imax maximum sample to look after

nbits number of digitiser bits

threshold is the distance from 0 that an adc value needs to be for it not to be saturated, as well as distance from 2^nbits

iunsat the returned last unsaturated sample

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 25 of file handle_saturation.c.

References bpm_error(), and bpm_warning().

Referenced by basic_stats(), ddc_sample_waveform(), and process_waveform().

5.9.1.10 EXTERN int downmix_waveform (double * *wf*, int *ns*, double *fs*, double *freq*, double *t0*, double ** *out*)

Performs the DDC on the input waveform

Parameters:

wf input waveform (with the pedestal substracted!)

ns number of samples in the waveform

fs sampling frequency

freq frequency of the signal

t0 sampling point

out complex output DDC waveform

Definition at line 21 of file downmix_waveform.c.

References bpm_error().

Referenced by ddc_sample_waveform(), and ddc_waveform().

5.9.1.11 EXTERN int ddc_gaussfilter_step (double ** *ddc*, int *ns*, double *fs*, int *istart*, int *istop*, double *tfilter*, double *filtBW*, double * *out*)

Performs one step in the gaussian filter

Parameters:

ddc the complex ddc wavform

ns number of samples

fs sampling frequency

istart starting sample for moving window

istop stop stample for moving window

tfilter filter time

filtBW filter bandwith

out a double[2] that will contain the resulting filtered Re and Im values at tfilter

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 25 of file ddc_gaussfilter_step.c.

References bpm_error(), bpm_warning(), and sample_to_time().

Referenced by ddc_gaussfilter(), and ddc_sample_waveform().

5.9.1.12 EXTERN int ddc_gaussfilter (double ** *ddc*, int *ns*, double *fs*, double *filtBW*, double *epsFilt*, double ** *out*)

Applies a gaussian filter to the total waveform with the given filter bandwidth and cut-off parameters

Parameters:

ddc complex double array with the downconverted waveform

ns number of samples

fs sampling frequency

filtBW filter bandwidth in MHz

epsFilt filter cutoff parameter

out complex double array with the filtered waveform

Returns:

BPM_SUCCESS upon succcess, BPM_FAILURE upon failure

Definition at line 23 of file ddc_gaussfilter.c.

References bpm_error(), ddc_gaussfilter_step(), sample_to_time(), and time_to_sample().

Referenced by ddc_waveform().

5.9.1.13 EXTERN int ddc_waveform (int * *wf*, int *ns*, int *nbits*, double *fs*, double *t0*, double *freq*, double *tdecay*, double *filtBW*, double *epsFilt*, double ** *out*)

Does the DDC of the full waveform and stores it into the ampwf and phasewf waveforms this routine calls the simple ddc(...) routine to do one step. Note that this one doesn't need t0 or t0Offset as it will scan through the entire waveform...

Parameters:

wf the waveform

ns the number of samples

nbits the number of digitiser bits

fs the sampling frequency

t0 the trigger time

freq the frequency of the waveform to downmix with

tdecay the decay time of the waveform

filtBW the gaussian filter bandwith

epsFilt the gaussian filter cut-off parameter

out contains the downconverted, filtered complex waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 39 of file ddc_waveform.c.

References alloc_complex_wave_double(), alloc_simple_wave_double(), bpm_error(), ddc_gaussfilter(), downmix_waveform(), free_complex_wave_double(), free_simple_wave_double(), and get_pedestal().

Referenced by process_waveform().

5.9.1.14 EXTERN int ddc_sample_waveform (int * *wf*, int *ns*, int *nbits*, double *fs*, double *t0*, double *t0Offset*, double *freq*, double *tdecay*, double *filtBW*, double *epsFilt*, double * *amp*, double * *phase*)

Does a quick DDC of the waveform and stores it into the ampwf and phasewf waveforms this routine calls the simple ddc(...) routine to do one step... the sampling point is determined by t0 + t0Offset

Parameters:

wf the waveform

ns the number of samples

nbits the number of digitiser bits

fs the sampling frequency

t0 the trigger time

t0Offset the sampling point

freq the frequency of the waveform to downmix with

tdecay the decay time of the waveform

filtBW the gaussian filter bandwith

epsFilt the gaussian filter cut-off parameter

amp amplitude at the sampling point

phase phase at the sampling point

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 40 of file ddc_sample_waveform.c.

References alloc_complex_wave_double(), bpm_error(), ddc_gaussfilter_step(), downmix_waveform(), free_complex_wave_double(), handle_saturation(), time_to_sample(), and usec.

Referenced by process_waveform().

5.9.1.15 EXTERN int get_pedestal (int * *wf*, int *ns*, int *range*, double * *offset*, double * *rms*)

Find the mean pedestal using the first 20 (or how ever many are required) sample values

Parameters:

wf a pointer to the waveform data

ns the number of samples in the waveform

range the maximum sample to go to average over

∗***offset*** returns the mean value of the samples, so voltage offset (pedestal value)

∗***rms*** returns the RMS on that

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 22 of file get_pedestal.c.

References bpm_error(), and bpm_warning().

Referenced by ddc_waveform(), fit_waveform(), get_t0(), and process_waveform().

5.9.1.16 EXTERN int basic_stats (int * *wf*, int *ns*, int *range*, int *nbits*, double * *offset*, double * *rms*, int * *max*, int * *min*, int * *unsat_sample*)

Find the mean pedestal using the first 20 (or how ever many are required) sample values

Parameters:

wf a pointer to the waveform data

ns the number of samples in the waveform

range the maximum sample to go to average over

nbits the number of digitiser bits

offset returns the mean value of the samples, so voltage offset (pedestal value)

rms returns the RMS on that

max returns max value of wf

min returns min value of wf

unsat_sample returns last unsaturated sample

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 26 of file basic_stats.c.

References bpm_error(), bpm_warning(), and handle_saturation().

5.9.1.17 EXTERN int int_to_double_waveform (double * *wf_double*, int * *wf_int*, int *ns*)

Cast int waveform values into double waveform values

Parameters:

**wf_double* waveform double

**wf_int* waveform int

ns the number of samples

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 20 of file int_to_double_waveform.c.

References bpm_error().

5.9.1.18 EXTERN int copy_waveform (double * *wf_dst*, double * *wf_src*, int *ns*)

Copies wf_src to wf_dst

Parameters:

wf_dst destination waveform

wf_src source waveform

ns the number of samples

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 20 of file copy_waveform.c.

References bpm_error().

5.9.1.19 EXTERN int mult_scalar_waveform (double * *wf*, int *ns*, double *mult*)

Multiply all values by a factor mult

Parameters:

wf the waveform

ns the number of samples

mult the factor to multiply all points in waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 20 of file mult_scalar_waveform.c.

References bpm_error().

5.9.1.20 EXTERN int mult_waveform (double * *wf1*, double * *wf2*, int *ns*)

Multiply all values by a factor mult

Parameters:

wf1 the waveform1, on return wf1 = wf1*wf2

wf2 the waveform2

ns the number of samples

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 20 of file mult_waveform.c.

References bpm_error().

5.9.1.21 EXTERN int get_t0 (int * *wf*, int *ns*, double *fs*, double * *t0*)

Finds the t0 value from a diode peak

Parameters:

wf a pointer to the waveform data

ns the number of samples in the waveform

fs sampling frequency

t0 returns t0 in usec

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 56 of file get_t0.c.

References bpm_error(), bpm_verbose, bpm_warning(), find_t0_endfit(), find_t0_startfit(), get_pedestal(), and nr_fit().

Referenced by fit_diodepulse().

5.9.1.22 EXTERN int time_to_sample (double *fs*, int *ns*, double *t*, int * *iS*)

Converts a time to a sample number, given the sampling frequency

Parameters:

fs sampling frequency

ns number of samples

t the queried time sample

iS the returned sample number

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 18 of file time_to_sample.c.

Referenced by ddc_gaussfilter(), and ddc_sample_waveform().

5.9.1.23 EXTERN int sample_to_time (double *fs*, int *ns*, int *iS*, double * *t*)

Converts a sample number to a time given the sampling frequency

Parameters:

fs sampling frequency

ns number of samples

t the queried sample

iS the returned sample time

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 18 of file sample_to_time.c.

Referenced by ddc_gaussfilter(), ddc_gaussfilter_step(), fcnwf(), fcnwfjac(), and process_waveform().

5.9.1.24 EXTERN int sample_to_freq (double *fs*, int *ns*, int *iS*, double * *f*)

This routine returns the frequency corresponding to the sample number, note that this routine is not aware of the nyquist bands, and just keeps on counting from 0 -> fs.

Parameters:

fs sampling frequency

ns number of samples

iS the queried sample to get the frequency of

f the returned frequency

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 20 of file sample_to_freq.c.

5.10 RF simulation routines

Files

- file **bpm_rf.h**
  libbpm rf simulation routines

- file **rf_addLO.c**
- file **rf_amplify.c**
- file **rf_butterworthbandpass.c**
- file **rf_butterworthhighpass.c**
- file **rf_butterworthlowpass.c**
- file **rf_complexFFT.c**
- file **rf_filter.c**
- file **rf_mixer.c**
- file **rf_rectify.c**
- file **rf_setup.c**

Functions

- EXTERN int **rf_setup** (int nsamples, double sfreq)
- EXTERN int **rf_rectify** (double **IF)
- EXTERN int **rf_filter** (double **RF, enum **rffiltertype_t** filttype, int nfiltpar, double *pars)
- EXTERN int **rf_butterworthlowpass** (double **RF, int order, double fc)
- EXTERN int **rf_butterworthbandpass** (double **RF, int order, double f0, double BW)
- EXTERN int **rf_butterworthhighpass** (double **RF, int order, double fc)
- EXTERN int **rf_complexFFT** (double **in, double **out, int dir)
- EXTERN int **rf_addLO** (double amp, double lofreq, enum **bpmphase_t** type, double phi0, double d_phi, double **LO)
- EXTERN int **rf_mixer** (double *RF_Re, double *RF_Im, double **LO, double *IF)
- EXTERN int **rf_amplify** (double *RF, double dB)

Variables

- EXTERN int **rf_nsamples**
- EXTERN double **rf_samplefreq**

5.10.1 Function Documentation

5.10.1.1 EXTERN int rf_setup (int *nsamples*, double *sfreq*)

Sets up the sampling of internal RF waveform representation

Parameters:

nsamples the number of samples
sfreq the internal sampling frequency

Returns:

BPM_SUCCESS

Definition at line 19 of file rf_setup.c.

References rf_nsamples, and rf_samplefreq.

5.10.1.2 EXTERN int rf_rectify (double ** *IF*)

Rectifies the given waveform

Parameters:

IF the complex waveform to rectify

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 16 of file rf_rectify.c.

References bpm_error(), and rf_nsamples.

Referenced by generate_diode().

5.10.1.3 EXTERN int rf_filter (double ** *RF*, enum rffiltertype_t *filttype*, int *nfiltpar*, double * *pars*)

Applies the filter to the RF waveform

Parameters:

RF the waveform
filttype the filter type
nfiltpar the number of filter parameters
pars the array of filter parameters

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 18 of file rf_filter.c.

References bpm_error(), butterworth_band_pass, butterworth_high_pass, butterworth_low_pass, nofilter, rf_butterworthbandpass(), rf_butterworthhighpass(), and rf_butterworthlowpass().

Referenced by generate_diode().

5.10.1.4 EXTERN int rf_butterworthlowpass (double ** *RF*, int *order*, double *fc*)

Apply a low pass Butterworth filter in frequency domain by taking the complex input waveform, and applying the frequency response

$$\frac{1}{1+\left(\frac{\nu}{\nu_c}\right)^{2n}}$$

Where $$ is the cut-off frequency and n the filter order.

Parameters:

RF the waveform

order The order of the filter

fc Filter cutoff frequency in MHz

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 24 of file rf_butterworthlowpass.c.

References alloc_complex_wave_double(), bpm_error(), free_complex_wave_double(), NR_FFTBACKWARD, NR_FFTFORWARD, rf_complexFFT(), rf_nsamples, and rf_samplefreq.

Referenced by rf_butterworthbandpass(), and rf_filter().

5.10.1.5 EXTERN int rf_butterworthbandpass (double ** *RF*, int *order*, double *f0*, double *BW*)

Apply a band pass Butterworth filter which is effectively a combination of a low and a high pass filters.

Parameters:

RF the waveform

order The order of the filter

f0 Central filter frequency in MHz

BW Bandwidth in MHz

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 21 of file rf_butterworthbandpass.c.

References bpm_error(), rf_butterworthhighpass(), and rf_butterworthlowpass().

Referenced by rf_filter().

5.10.1.6 EXTERN int rf_butterworthhighpass (double ** *RF*, int *order*, double *fc*)

Apply a high pass Butterworth filter in frequency domain by taking the complex input waveform, and applying the frequency response

$$\frac{1}{1+\left(\frac{\nu_c}{\nu}\right)^{2n}}$$

Where $$ is the cut-off frequency and n the filter order.

Parameters:

RF the waveform

order The order of the filter

fc Filter cutoff frequency in MHz

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 24 of file rf_butterworthhighpass.c.

References alloc_complex_wave_double(), bpm_error(), free_complex_wave_double(), NR_FFTBACKWARD, NR_FFTFORWARD, rf_complexFFT(), rf_nsamples, and rf_samplefreq.

Referenced by rf_butterworthbandpass(), and rf_filter().

5.10.1.7 EXTERN int rf_complexFFT (double ** *in*, double ** *out*, int *dir*)

Perform a complex FFT on the input waveform

Parameters:

in the input waveform

out the output FFT

dir forward or inverse?

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 19 of file rf_complexFFT.c.

References bpm_error(), nr_four1(), and rf_nsamples.

Referenced by rf_butterworthhighpass(), and rf_butterworthlowpass().

5.10.1.8 EXTERN int rf_addLO (double *amp*, double *lofreq*, enum bpmphase_t *type*, double *phi0*, double *d_phi*, double ** *LO*)

Adds an LO signal to the waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 15 of file rf_addLO.c.

References bpm_error(), locked, nr_rangauss(), nr_ranuniform(), rf_nsamples, and rf_samplefreq.

5.10.1.9 EXTERN int rf_mixer (double * *RF_Re*, double * *RF_Im*, double ** *LO*, double * *IF*)

Mixes the RF and the LO to produce the IF

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 13 of file rf_mixer.c.

References bpm_error(), and rf_nsamples.

5.10.1.10 EXTERN int rf_amplify (double * *RF*, double *dB*)

Amplifies the signal by the level dB. The voltage gain is calculated:

gain = sqrt(10^(db/20))

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 15 of file rf_amplify.c.

References bpm_error(), and rf_nsamples.

5.10.2 Variable Documentation

5.10.2.1 EXTERN int rf_nsamples

Numer of samples in the rf waveform representations, default value is 2^16 = 65536

Definition at line 43 of file bpm_rf.h.

Referenced by add_amplnoise(), add_wave(), add_waveforms(), digitise(), generate_diode(), generate_dipole(), generate_monopole(), get_amplitude(), get_complex_from_AmpPhi(), get_complex_from_ReIm(), get_imaginary_part(), get_phase(), get_real_part(), reset_complex_wave(), rf_addLO(), rf_amplify(), rf_butterworthhighpass(), rf_butterworthlowpass(), rf_complexFFT(), rf_mixer(), rf_rectify(), and rf_setup().

5.10.2.2 EXTERN double rf_samplefreq

Effective sampling frequency for the rf waveform representations, default value is 20 GHz

Definition at line 49 of file bpm_rf.h.

Referenced by add_excitation(), add_wave(), digitise(), rf_addLO(), rf_butterworthhighpass(), rf_butterworthlowpass(), and rf_setup().

5.11 BPM signal simulation routines

Files

- file **add_amplnoise.c**
- file **add_excitation.c**
- file **add_wave.c**

- file **bpm_simulation.h**

  libbpm waveform simulation routines

- file **digitise.c**
- file **generate_diode.c**
- file **generate_dipole.c**
- file **generate_monopole.c**
- file **generate_noise.c**
- file **get_amplitude.c**
- file **get_complex_from_AmpPhi.c**
- file **get_complex_from_ReIm.c**
- file **get_dipole_amp.c**
- file **get_dipole_response.c**
- file **get_imaginary_part.c**
- file **get_monopole_amp.c**
- file **get_monopole_response.c**
- file **get_phase.c**
- file **get_real_part.c**
- file **reset_complex_wave.c**
- file **reset_simple_wave.c**
- file **simple_tone.c**
- file **simple_wave.c**

Functions

- EXTERN int **generate_monopole** (**bpmconf_t** *, **beamconf_t** *, **bpmsignal_t** *)
- EXTERN int **generate_dipole** (**bpmconf_t** *, **beamconf_t** *, **bpmsignal_t** *)
- EXTERN int **generate_diode** (**bpmconf_t** *, **beamconf_t** *, **bpmsignal_t** *)
- EXTERN int **get_monopole_response** (double bunchcharge, double chargesens, double arrivaltime, double cavityfreq, double *amp, double *phase)
- EXTERN int **get_dipole_response** (double bunchcharge, double chargesens, double pos, double possens, double tilt, double tiltsens, double arrivaltime, double cavityfreq, double *amp, double *phase)
- EXTERN int **get_dipole_amp** (double bunchcharge, double bunchlength, double pos, double possens, double slope, double slopesens, double tilt, double tiltsens, double *amp, double *phase)
- EXTERN int **get_monopole_amp** (double bunchcharge, double bunchlength, double chargesens, double *amp, double *phase)
- EXTERN int **add_excitation** (double ttrig, double *RF)
- EXTERN int **simple_wave** (double amp, double phase, double ttrig, double freq, double tdecay, double ped, double ampnoise, double phasenoise, double fs, int nbits, int *wf, int ns)
- EXTERN int **simple_tone** (double amp, double phase, double freq, double ped, double ampnoise, double phasenoise, double fs, int nbits, int *wf, int ns)
- EXTERN int **add_wave** (double amp, double phase, double freq, double ttrig, double tdecay, double **RF)
- EXTERN int **add_waveforms** (double *RF, double *RFadd, double factor)
- EXTERN int **reset_complex_wave** (double **RF)
- EXTERN int **reset_simple_wave** (int ns, double *wf)
- EXTERN int **get_real_part** (double **RF, double *wf)
- EXTERN int **get_imaginary_part** (double **RF, double *wf)

- EXTERN int **get_amplitude** (double **RF, double *wf)
- EXTERN int **get_phase** (double **RF, double *wf)
- EXTERN int **get_complex_from_ReIm** (double *RF_Re, double *RF_Im, double **RF)
- EXTERN int **get_complex_from_AmpPhi** (double *Amp, double *Phi, double **RF)
- EXTERN int **add_amplnoise** (double amplnoise, double *IF_Re, double *IF_Im)
- EXTERN int **digitise** (double *IF, int nbits, double fs, double range_min, double range_max, int ns, int *wf)

5.11.1 Function Documentation

5.11.1.1 EXTERN int generate_monopole (bpmconf_t * *bpm*, beamconf_t * *beam*, bpmsignal_t * *sig*)

Generate monopole waveform

Definition at line 13 of file generate_monopole.c.

References add_wave(), alloc_complex_wave_double(), alloc_simple_wave_double(), alloc_simple_wave_int(), beamconf::arrival_time, bpm_error(), bpmconf::cav_chargesens, bpmconf::cav_decaytime, bpmconf::cav_freq, beamconf::charge, bpmconf::digi_ampnoise, bpmconf::digi_nbits, bpmconf::digi_nsamples, bpmconf::digi_trigtimeoffset, free_complex_wave_double(), free_simple_wave_double(), get_monopole_response(), bpmsignal::ns, rf_nsamples, and bpmsignal::wf.

5.11.1.2 EXTERN int generate_dipole (bpmconf_t * *bpm*, beamconf_t * *beam*, bpmsignal_t * *sig*)

Generate dipole waveform

Definition at line 14 of file generate_dipole.c.

References add_wave(), alloc_complex_wave_double(), alloc_simple_wave_double(), alloc_simple_wave_int(), beamconf::arrival_time, bpm_error(), beamconf::bpmhit, beamconf::bpmtilt, bpmconf::cav_chargesens, bpmconf::cav_decaytime, bpmconf::cav_freq, bpmconf::cav_polarisation, bpmconf::cav_possens, bpmconf::cav_tiltsens, beamconf::charge, bpmconf::digi_ampnoise, bpmconf::digi_nbits, bpmconf::digi_nsamples, bpmconf::digi_trigtimeoffset, free_complex_wave_double(), free_simple_wave_double(), get_dipole_response(), horiz, bpmsignal::ns, rf_nsamples, and bpmsignal::wf.

5.11.1.3 EXTERN int generate_diode (bpmconf_t * *bpm*, beamconf_t * *beam*, bpmsignal_t * *sig*)

Generate a diode waveform using the given bpm parameters. Essentially, a reference waveform is generated and then rectified to produce the trigger pulse.

Parameters:

bpm Structure containing the BPM info

beam Structure containing the beam hit info

sig Structure for where to place the digitised waveform

Definition at line 21 of file generate_diode.c.

References add_wave(), alloc_complex_wave_double(), alloc_simple_wave_int(), beamconf::arrival_time, bpm_error(), bpmconf::cav_chargesens, bpmconf::cav_decaytime,

bpmconf::cav_freq, beamconf::charge, bpmconf::digi_ampnoise, bpmconf::digi_nbits, bpmconf::digi_nsamples, bpmconf::digi_trigtimeoffset, free_complex_wave_double(), get_monopole_response(), bpmsignal::ns, rf_filter(), bpmconf::rf_filterpars, bpmconf::rf_filtertype, bpmconf::rf_nfiltpars, rf_nsamples, rf_rectify(), and bpmsignal::wf.

5.11.1.4 EXTERN int get_monopole_response (double *bunchcharge*, double *chargesens*, double *arrivaltime*, double *cavityfreq*, double * *amp*, double * *phase*)

Calculate the response of a monopole signal given an incoming bunch

NOTE: Still have phase questions!

Parameters:

bunchcharge The charge of the bunch (in nC)

chargesens The charge sensitivty of the BPM

arrivaltime The beam arrival time

cavityfreq The frequency of the cavity

amp the amplitude of the waveform at the arrival time

phase the phase of the waveform at the arrival time

Definition at line 20 of file get_monopole_response.c.

Referenced by generate_diode(), and generate_monopole().

5.11.1.5 EXTERN int get_dipole_response (double *bunchcharge*, double *chargesens*, double *pos*, double *possens*, double *tilt*, double *tiltsens*, double *arrivaltime*, double *cavityfreq*, double * *amp*, double * *phase*)

Calculate the response of a dipole signal given an incoming bunch

NOTE: Still have phase questions! Alex**May need to include the bunch length**

Parameters:

bunchcharge The charge of the bunch (in nC)

chargesens The charge sensitivty of the BPM **Alex**remove this**

pos The position of the beam (in um)

possens The position sensitivty of the BPM **Alex**mV/nC/mm**

tilt The tilt of the beam (in urad)

tiltsens The tilt sensitivty of the BPM **Alex**mV/nC/mrad**

arrivaltime The beam arrival time

cavityfreq The frequency of the cavity

amp the amplitude of the waveform at the arrival time

phase the phase of the waveform at the arrival time

Definition at line 25 of file get_dipole_response.c.

Referenced by generate_dipole().

5.11.1.6 EXTERN int get_dipole_amp (double *bunchcharge*, double *bunchlength*, double *pos*, double *possens*, double *slope*, double *slopesens*, double *tilt*, double *tiltsens*, double * *amp*, double * *phase*)

Calculate the response of a dipole signal given an incoming bunch

Parameters:

bunchcharge The charge of the bunch (in nC)

bunchlength The length of the bunch (in mm)

pos The position of the beam (in mm)

possens The position sensitivty of the BPM (in V/nC/mm)

slope The slope of the beam (in rad)

slopesens The slope sensitivity of the BPM (in V/nC/urad)

tilt The tilt of the bunch (in urad)

tiltsens The tilt sensitivty of the BPM (V/nC/urad)

amp the amplitude of the waveform at the arrival time

phase the phase of the waveform at the arrival time

Definition at line 23 of file get_dipole_amp.c.

5.11.1.7 EXTERN int get_monopole_amp (double *bunchcharge*, double *bunchlength*, double *chargesens*, double * *amp*, double * *phase*)

Calculate the response of a dipole signal given an incoming bunch

Parameters:

bunchcharge The charge of the bunch (in nC)

bunchlength The length of the bunch (in mm)

chargesens The charge sensitivty of the BPM (V/nC)

amp the amplitude of the waveform at the arrival time

phase the phase of the waveform at the arrival time

Definition at line 18 of file get_monopole_amp.c.

5.11.1.8 EXTERN int add_excitation (double *ttrig*, double * *RF*)

Generates a step to excite the resonator

Parameters:

ttrig the trigger time

RF waveform

Returns:

BPM_SUCCES upon succes, BPM_FAILURE upon failure

Definition at line 19 of file add_excitation.c.

References bpm_error(), and rf_samplefreq.

5.11.1.9 EXTERN int simple_wave (double *amp*, double *phase*, double *ttrig*, double *freq*, double *tdecay*, double *ped*, double *ampnoise*, double *phasenoise*, double *fs*, int *nbits*, int * *wf*, int *ns*)

Just fills an array of integers straight from the parameters given, this routine is a little tool to quickly generate digitised decaying waveforms without much RF hassle. The routine overwrites the wf array, so doesn't add the values...

The waveform shape is

$$Ae^{-(t-t0)/\tau}sin(2\pi\nu(t-t0)+\phi)+offset$$

with added phasenoise to the phase and amplitude noise to the offset. The user needs to give the nbits in the ADC as well to enable the routine to cut off the sample and have saturation...

Parameters:

amp amplitude in ADC channels

phase phase of waveform

ttrig trigger time of waveform (t0)

freq frequency of waveform

tdecay decay time of waveform

ped offset of waveform in ADC channels

ampnoise amplitude noise in ADC channels

phasenoise phase noise

fs sampling frequency

nbits number of bits in the ADC

∗***wf*** the returned waveform

ns the number of samples to fill

Returns:

BPM_SUCCESS upon success, BPM_ERROR upon error

Definition at line 39 of file simple_wave.c.

References bpm_error(), nr_rangauss(), and usec.

5.11.1.10 EXTERN int simple_tone (double *amp*, double *phase*, double *freq*, double *ped*, double *ampnoise*, double *phasenoise*, double *fs*, int *nbits*, int * *wf*, int *ns*)

Just fills an array of integers straight from the parameters given, this routine is a little tool to quickly generate digitised tone signals without much RF hassle. The routine overwrites the wf array, so doesn't add the values...

The waveform shape is

$$Asin(2\pi\nu t+\phi)+offset$$

with added phasenoise to the phase and amplitude noise to the offset. The user needs to give the nbits in the ADC as well to enable the routine to cut off the sample and have saturation...

Parameters:

amp amplitude in ADC channels

phase phase of the tone

freq frequency of tone
ped offset of tone in ADC channels
ampnoise amplitude noise in ADC channels
phasenoise phase noise
fs sampling frequency
nbits number of bits in the ADC
∗***wf*** the returned waveform
ns the number of samples to fill

Returns:

BPM_SUCCESS upon success, BPM_ERROR upon error

Definition at line 37 of file simple_tone.c.

References bpm_error(), nr_rangauss(), and usec.

5.11.1.11 EXTERN int add_wave (double *amp*, double *phase*, double *freq*, double *ttrig*, double *tdecay*, double ∗∗ *RF*)

Builds up an RF waveform from the given amp, phase frequency and tdecay... uses the rf_ parameters which dictate the internal representation of the RF wave

Definition at line 15 of file add_wave.c.

References bpm_error(), rf_nsamples, and rf_samplefreq.

Referenced by generate_diode(), generate_dipole(), and generate_monopole().

5.11.1.12 EXTERN int add_waveforms (double ∗ *RF*, double ∗ *RFadd*, double *factor*)

Definition at line 8 of file add_waveforms.c.

References bpm_error(), and rf_nsamples.

5.11.1.13 EXTERN int reset_complex_wave (double ∗∗ *RF*)

Resets the array elements of RF to 0+i0

Definition at line 12 of file reset_complex_wave.c.

References bpm_error(), and rf_nsamples.

5.11.1.14 EXTERN int reset_simple_wave (int *ns*, double ∗ *wf*)

Resets the array elements of RF to 0

Definition at line 12 of file reset_simple_wave.c.

References bpm_error().

5.11.1.15 EXTERN int get_real_part (double ∗∗ *RF*, double ∗ *wf*)

Returns the real part of the array elements

Definition at line 12 of file get_real_part.c.

References bpm_error(), and rf_nsamples.

5.11.1.16 EXTERN int get_imaginary_part (double ** *RF*, double * *wf*)

Returns the real part of the array elements

Definition at line 12 of file get_imaginary_part.c.

References bpm_error(), and rf_nsamples.

5.11.1.17 EXTERN int get_amplitude (double ** *RF*, double * *wf*)

Returns the amplitudes of the complex array

Definition at line 12 of file get_amplitude.c.

References bpm_error(), and rf_nsamples.

5.11.1.18 EXTERN int get_phase (double ** *RF*, double * *wf*)

Returns the phases of the complex array

Definition at line 12 of file get_phase.c.

References bpm_error(), and rf_nsamples.

5.11.1.19 EXTERN int get_complex_from_ReIm (double * *RF_Re*, double * *RF_Im*, double ** *RF*)

Commbines real and imaginary parts into a complex array

Definition at line 12 of file get_complex_from_ReIm.c.

References bpm_error(), and rf_nsamples.

5.11.1.20 EXTERN int get_complex_from_AmpPhi (double * *Amp*, double * *Phi*, double ** *RF*)

Commbines amplitude and phase into a complex array

Definition at line 12 of file get_complex_from_AmpPhi.c.

References bpm_error(), and rf_nsamples.

5.11.1.21 EXTERN int add_amplnoise (double *amplnoise*, double * *IF_Re*, double * *IF_Im*)

Add the given amount of amplitude noise to the array

Parameters:

amplnoise The amplitude noise to add to the waveform (in Volts)

IF_Re The real part of the waveform

IF_Im The imaginary part of the waveform

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 20 of file add_amplnoise.c.

References bpm_error(), nr_rangauss(), nr_ranuniform(), and rf_nsamples.

5.11.1.22 EXTERN int digitise (double * *IF*, int *nbits*, double *fs*, double *range_min*, double *range_max*, int *ns*, int * *wf*)

Digitises the waveform

Parameters:

IF The input waveofrm to digitse

nbits The number of bits of the ADC

fs the sampling frequency

range_min the minimum voltage and

range_max the maximum voltage the ADC can process

ns number of samples in the sampled waveform

wf sampled waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 23 of file digitise.c.

References bpm_error(), nr_rangauss(), rf_nsamples, and rf_samplefreq.

6 libbpm Data Structure Documentation

6.1 _gsl_matrix_view Struct Reference

Collaboration diagram for _gsl_matrix_view:

6.1.1 Detailed Description

Definition at line 166 of file bpm_nr.h.

Data Fields

- **gsl_matrix matrix**

The documentation for this struct was generated from the following file:

- bpmnr/**bpm_nr.h**

6.2 _gsl_vector_const_view Struct Reference

Collaboration diagram for _gsl_vector_const_view:

6.2.1 Detailed Description

Definition at line 194 of file bpm_nr.h.

Data Fields

- **gsl_vector vector**

The documentation for this struct was generated from the following file:

- bpmnr/**bpm_nr.h**

6.3 _gsl_vector_view Struct Reference

Collaboration diagram for _gsl_vector_view:

6.3.1 Detailed Description

Definition at line 186 of file bpm_nr.h.

Data Fields

- **gsl_vector vector**

The documentation for this struct was generated from the following file:

- bpmnr/**bpm_nr.h**

6.4 beamconf Struct Reference

`#include <bpm_interface.h>`

6.4.1 Detailed Description

This structure contains the beam information at a certain point of the orbit.

Definition at line 187 of file bpm_interface.h.

Data Fields

- double **energy**
- double **sig_energy**
- double **charge**
- double **sig_charge**
- double **arrival_time**
- double **beampos** [2]
- double **beamslope** [2]
- double **bpmhit** [3]
- double **bpmtilt** [3]

6.4.2 Field Documentation

6.4.2.1 double beamconf::energy

average beam energy (in GeV)

Definition at line 188 of file bpm_interface.h.

6.4.2.2 double beamconf::sig_energy

energy spread (sigma)

Definition at line 189 of file bpm_interface.h.

6.4.2.3 double beamconf::charge

bunch charge (in nC)

Definition at line 190 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), and generate_monopole().

6.4.2.4 double beamconf::sig_charge

charge spread (sigma)

Definition at line 191 of file bpm_interface.h.

6.4.2.5 double beamconf::arrival_time

arrival time of bunch, in (usec)

Definition at line 193 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), and generate_monopole().

6.4.2.6 double beamconf::beampos[2]

the beam position x,y at the bpm coo

Definition at line 195 of file bpm_interface.h.

Referenced by generate_bpm_orbit(), generate_corr_scan(), and generate_mover_scan().

6.4.2.7 double beamconf::beamslope[2]

the beam slope x',y' at the bpm coo

Definition at line 196 of file bpm_interface.h.

Referenced by generate_bpm_orbit(), and generate_corr_scan().

6.4.2.8 double beamconf::bpmhit[3]

where the beam hits the BPM in the BPM local co

Definition at line 198 of file bpm_interface.h.

Referenced by calibrate(), generate_bpm_orbit(), generate_dipole(), and setup_calibration().

6.4.2.9 double beamconf::bpmtilt[3]

tilt of the beam through the BPM in the BPM local co

Definition at line 199 of file bpm_interface.h.

Referenced by generate_bpm_orbit(), and generate_dipole().

The documentation for this struct was generated from the following file:

- bpminterface/**bpm_interface.h**

6.5 bpmcalib Struct Reference

`#include <bpm_interface.h>`

6.5.1 Detailed Description

A structure containing the calibration information

Definition at line 135 of file bpm_interface.h.

Data Fields

- double **freq**
- double **tdecay**
- double **ddcfiltBW**
- double **ddcepsFilt**
- double **t0Offset**
- double **IQphase**
- double **posscale**
- double **slopescale**

6.5.2 Field Documentation

6.5.2.1 double bpmcalib::freq

frequency of downmixed waveform (MHz)

Definition at line 136 of file bpm_interface.h.

Referenced by load_calibration(), and process_waveform().

6.5.2.2 double bpmcalib::tdecay

decay time (usec)

Definition at line 137 of file bpm_interface.h.

Referenced by load_calibration(), and process_waveform().

6.5.2.3 double bpmcalib::ddcfiltBW

ddc filter bandwidth in MHz

Definition at line 138 of file bpm_interface.h.

Referenced by load_calibration(), and process_waveform().

6.5.2.4 double bpmcalib::ddcepsFilt

ddc epsilon filter

Definition at line 139 of file bpm_interface.h.

Referenced by load_calibration(), and process_waveform().

6.5.2.5 double bpmcalib::t0Offset

always have offset from t0 for sampling !!!

Definition at line 140 of file bpm_interface.h.

Referenced by load_calibration(), and process_waveform().

6.5.2.6 double bpmcalib::IQphase

processed IQ phase

Definition at line 141 of file bpm_interface.h.

Referenced by calibrate(), load_calibration(), and process_dipole().

6.5.2.7 double bpmcalib::posscale

processed position scale

Definition at line 142 of file bpm_interface.h.

Referenced by load_calibration(), and process_dipole().

6.5.2.8 double bpmcalib::slopescale

processed slope scale

Definition at line 143 of file bpm_interface.h.

Referenced by load_calibration(), and process_dipole().

The documentation for this struct was generated from the following file:

- bpminterface/**bpm_interface.h**

6.6 bpmconf Struct Reference

`#include <bpm_interface.h>`

6.6.1 Detailed Description

Structure containing the BPM configuration

Definition at line 85 of file bpm_interface.h.

Data Fields

- char **name** [20]
- enum **bpmtype_t cav_type**

- enum **bpmpol_t cav_polarisation**
- enum **bpmphase_t cav_phasetype**
- double **cav_freq**
- double **cav_decaytime**
- double **cav_phase**
- double **cav_iqrotation**
- double **cav_chargesens**
- double **cav_possens**
- double **cav_tiltsens**
- double **rf_LOfreq**
- enum **rffiltertype_t rf_filtertype**
- int **rf_nfiltpars**
- double * **rf_filterpars**
- double **rf_gain**
- double **digi_trigtimeoffset**
- double **digi_freq**
- int **digi_nbits**
- int **digi_nsamples**
- double **digi_ampnoise**
- int **digi_voltageoffset**
- double **digi_phasenoise**
- double **geom_pos** [3]
- double **geom_tilt** [3]
- int **ref_idx**
- int **diode_idx**

6.6.2 Field Documentation

6.6.2.1 char bpmconf::name[20]

a BPM should have a name

Definition at line 86 of file bpm_interface.h.

Referenced by process_diode(), process_dipole(), and process_waveform().

6.6.2.2 enum bpmtype_t bpmconf::cav_type

BPM type

Definition at line 88 of file bpm_interface.h.

Referenced by process_diode(), and process_waveform().

6.6.2.3 enum bpmpol_t bpmconf::cav_polarisation

BPM polarisation

Definition at line 89 of file bpm_interface.h.

Referenced by calibrate(), and generate_dipole().

6.6.2.4 enum bpmphase_t bpmconf::cav_phasetype

BPM phase type

Definition at line 90 of file bpm_interface.h.

6.6.2.5 double bpmconf::cav_freq

cavity freq (MHz)

Definition at line 92 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), generate_monopole(), and process_waveform().

6.6.2.6 double bpmconf::cav_decaytime

cavity decay time (microsec)

Definition at line 93 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), generate_monopole(), and process_waveform().

6.6.2.7 double bpmconf::cav_phase

phase advance wrt. reference (fixed or random)

Definition at line 94 of file bpm_interface.h.

6.6.2.8 double bpmconf::cav_iqrotation

cavity IQ rotation

Definition at line 95 of file bpm_interface.h.

6.6.2.9 double bpmconf::cav_chargesens

charge sensitivity (volt/nC)

Definition at line 96 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), and generate_monopole().

6.6.2.10 double bpmconf::cav_possens

position sensitivity at 1.6nC charge (volt/micron)

Definition at line 97 of file bpm_interface.h.

Referenced by generate_dipole().

6.6.2.11 double bpmconf::cav_tiltsens

tilt sensitivity at 1.6nC charge (volt/micron)

Definition at line 98 of file bpm_interface.h.

Referenced by generate_dipole().

6.6.2.12 double bpmconf::rf_LOfreq

LO frequency to mix down with (in MHz)

Definition at line 100 of file bpm_interface.h.

Referenced by process_waveform().

6.6.2.13 enum rffiltertype_t bpmconf::rf_filtertype

RF filter type

Definition at line 101 of file bpm_interface.h.

Referenced by generate_diode().

6.6.2.14 double* bpmconf::rf_filterpars

RF filter parameters

Definition at line 103 of file bpm_interface.h.

Referenced by generate_diode().

6.6.2.15 double bpmconf::rf_gain

Gain of the electronics

Definition at line 104 of file bpm_interface.h.

6.6.2.16 double bpmconf::digi_trigtimeoffset

time (usec) to offset bunch arrival times by

Definition at line 106 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), and generate_monopole().

6.6.2.17 double bpmconf::digi_freq

digitization frequency (MHz)

Definition at line 107 of file bpm_interface.h.

Referenced by process_diode(), and process_waveform().

6.6.2.18 int bpmconf::digi_nbits

number of bits in ADC for digitisation

Definition at line 108 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), generate_monopole(), and process_waveform().

6.6.2.19 int bpmconf::digi_nsamples

number of samples in ADC digitisation

Definition at line 109 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), generate_monopole(), process_diode(), and process_waveform().

6.6.2.20 double bpmconf::digi_ampnoise

amplitude noise in ADC channels (pedestal width)

Definition at line 110 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), and generate_monopole().

6.6.2.21 int bpmconf::digi_voltageoffset

voltage offset (pedestal position) in counts

Definition at line 111 of file bpm_interface.h.

6.6.2.22 double bpmconf::digi_phasenoise

phase noise

Definition at line 112 of file bpm_interface.h.

6.6.2.23 double bpmconf::geom_pos[3]

position of the BPM in the beamline

Definition at line 115 of file bpm_interface.h.

Referenced by generate_bpm_orbit(), and generate_corr_scan().

6.6.2.24 double bpmconf::geom_tilt[3]

tilt of the BPM (0: theta/dx/y', 1: phi/dy/x', 2: roll)

Definition at line 116 of file bpm_interface.h.

Referenced by generate_bpm_orbit().

6.6.2.25 int bpmconf::ref_idx

reference cavity index for this BPM

Definition at line 118 of file bpm_interface.h.

6.6.2.26 int bpmconf::diode_idx

reference diode index for this BPM

Definition at line 119 of file bpm_interface.h.

The documentation for this struct was generated from the following file:

- bpminterface/**bpm_interface.h**

6.7 bpmproc Struct Reference

`#include <bpm_interface.h>`

6.7.1 Detailed Description

A structure containing the processed waveform information

Definition at line 149 of file bpm_interface.h.

Data Fields

- double **ampnoise**
- double **voltageoffset**
- double **t0**
- double ** **ddcwf**
- double ** **fftwf**
- int **fft_success**
- double **fft_freq**
- double **fft_tdecay**
- int **ddc_success**
- double **ddc_Q**
- double **ddc_I**
- double **ddc_amp**
- double **ddc_phase**
- double **ddc_tdecay**
- double **ddc_pos**
- double **ddc_slope**
- int **fit_success**
- double **fit_Q**
- double **fit_I**
- double **fit_amp**
- double **fit_phase**
- double **fit_freq**
- double **fit_tdecay**
- double **fit_pos**
- double **fit_slope**

6.7.2 Field Documentation

6.7.2.1 double bpmproc::ampnoise

calculated (processed) amplitude noise

Definition at line 151 of file bpm_interface.h.

Referenced by process_waveform().

6.7.2.2 double bpmproc::voltageoffset

calculated voltage offset

Definition at line 152 of file bpm_interface.h.

Referenced by process_waveform().

6.7.2.3 double bpmproc::t0

trigger t0 signal

Definition at line 154 of file bpm_interface.h.

Referenced by process_diode(), and process_waveform().

6.7.2.4 double** bpmproc::ddcwf

The digially down converted waveform

Definition at line 156 of file bpm_interface.h.

Referenced by process_waveform().

6.7.2.5 double** bpmproc::fftwf

The fourier transform of the waveform

Definition at line 157 of file bpm_interface.h.

Referenced by process_waveform().

6.7.2.6 int bpmproc::fft_success

do we have proper fft info ?

Definition at line 159 of file bpm_interface.h.

Referenced by process_waveform().

6.7.2.7 double bpmproc::fft_freq

frequency obtained from fft (MHz)

Definition at line 160 of file bpm_interface.h.

Referenced by process_waveform().

6.7.2.8 double bpmproc::fft_tdecay

decay time obtained from fft (usec)

Definition at line 161 of file bpm_interface.h.

Referenced by process_waveform().

6.7.2.9 int bpmproc::ddc_success

do we have proper ddc info ?

Definition at line 163 of file bpm_interface.h.

Referenced by process_dipole(), and process_waveform().

6.7.2.10 double bpmproc::ddc_Q

ddc Q value

Definition at line 164 of file bpm_interface.h.

Referenced by calibrate(), and process_dipole().

6.7.2.11 double bpmproc::ddc_I

ddc I value

Definition at line 165 of file bpm_interface.h.

Referenced by process_dipole().

6.7.2.12 double bpmproc::ddc_amp

downconverted amplitude

Definition at line 166 of file bpm_interface.h.

Referenced by process_dipole(), and process_waveform().

6.7.2.13 double bpmproc::ddc_phase

downconverted phase

Definition at line 167 of file bpm_interface.h.

Referenced by process_dipole(), and process_waveform().

6.7.2.14 double bpmproc::ddc_tdecay

downconverted decay time of waveform

Definition at line 168 of file bpm_interface.h.

6.7.2.15 double bpmproc::ddc_pos

calculated position from ddc

Definition at line 169 of file bpm_interface.h.

Referenced by ana_compute_residual(), and process_dipole().

6.7.2.16 double bpmproc::ddc_slope

calculated slope from ddc

Definition at line 170 of file bpm_interface.h.

Referenced by process_dipole().

6.7.2.17 int bpmproc::fit_success

do we have proper fit info ?

Definition at line 172 of file bpm_interface.h.

Referenced by process_dipole(), and process_waveform().

6.7.2.18 double bpmproc::fit_Q

fit Q value

Definition at line 173 of file bpm_interface.h.

Referenced by process_dipole().

6.7.2.19 double bpmproc::fit_I

fit I value

Definition at line 174 of file bpm_interface.h.

Referenced by process_dipole().

6.7.2.20 double bpmproc::fit_amp

fitted amplitude

Definition at line 175 of file bpm_interface.h.

Referenced by process_dipole(), and process_waveform().

6.7.2.21 double bpmproc::fit_phase

fitted phase

Definition at line 176 of file bpm_interface.h.

Referenced by process_dipole(), and process_waveform().

6.7.2.22 double bpmproc::fit_freq

fitted frequency (MHz)

Definition at line 177 of file bpm_interface.h.

Referenced by process_waveform().

6.7.2.23 double bpmproc::fit_tdecay

fitted decay time of waveform (usec)

Definition at line 178 of file bpm_interface.h.

Referenced by process_waveform().

6.7.2.24 double bpmproc::fit_pos

calculated position from fit

Definition at line 179 of file bpm_interface.h.

Referenced by process_dipole().

6.7.2.25 double bpmproc::fit_slope

calculated slope from fit

Definition at line 180 of file bpm_interface.h.

Referenced by process_dipole().

The documentation for this struct was generated from the following file:

- bpminterface/**bpm_interface.h**

6.8 bpmsignal Struct Reference

`#include <bpm_interface.h>`

6.8.1 Detailed Description

A structure holding the BPM signal

Definition at line 126 of file bpm_interface.h.

Data Fields

- int * **wf**
- int **ns**

6.8.2 Field Documentation

6.8.2.1 int* bpmsignal::wf

BPM signal

Definition at line 127 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), generate_monopole(), process_diode(), and process_waveform().

6.8.2.2 int bpmsignal::ns

Number of samples for the waveform (just in case)

Definition at line 128 of file bpm_interface.h.

Referenced by generate_diode(), generate_dipole(), generate_monopole(), and save_signals().

The documentation for this struct was generated from the following file:

- bpminterface/**bpm_interface.h**

6.9 complex_t Struct Reference

`#include <bpm_nr.h>`

6.9.1 Detailed Description

Structure and typedef for complex numbers used in the bpmdsp module

Definition at line 206 of file bpm_nr.h.

Data Fields

- double **re**

- double **im**

The documentation for this struct was generated from the following file:

- bpmnr/**bpm_nr.h**

6.10 filter_t Struct Reference

`#include <bpm_dsp.h>`

Collaboration diagram for filter_t:

6.10.1 Detailed Description

The filter structure.

Definition at line 75 of file bpm_dsp.h.

Data Fields

- char **name** [80]
- unsigned int **options**
- int **order**
- double **fs**
- double **f1**
- double **f2**
- double **alpha1**
- double **alpha2**
- double **w_alpha1**
- double **w_alpha2**
- double **cheb_ripple**
- double **Q**
- **complex_t dc_gain**
- **complex_t fc_gain**
- **complex_t hf_gain**
- double **gain**
- **filterrep_t** * **cplane**
- int **IsFIR**
- int **nxc**
- double **xc** [MAXPZ+1]
- int **nyc**
- double **yc** [MAXPZ+1]
- double **xv** [MAXPZ+1]
- double **yv** [MAXPZ+1]
- int **ns**
- double * **wfbuffer**

6.10.2 Field Documentation

6.10.2.1 char filter_t::name[80]

The filter's name

Definition at line 76 of file bpm_dsp.h.

Referenced by create_filter(), and print_filter().

6.10.2.2 unsigned int filter_t::options

type and option bits for filter

Definition at line 78 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), create_filter(), create_resonator_representation(), create_splane_representation(), normalise_filter(), and zplane_transform().

6.10.2.3 int filter_t::order

filter order

Definition at line 79 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

6.10.2.4 double filter_t::fs

sampling frequency

Definition at line 81 of file bpm_dsp.h.

Referenced by create_filter().

6.10.2.5 double filter_t::f1

first frequency (left edge for bandpass/stop)

Definition at line 82 of file bpm_dsp.h.

Referenced by create_filter().

6.10.2.6 double filter_t::f2

right edge for bandpass/stop (undef for low/highpass)

Definition at line 83 of file bpm_dsp.h.

Referenced by create_filter().

6.10.2.7 double filter_t::alpha1

rescaled f1

Definition at line 85 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), create_filter(), and create_resonator_representation().

6.10.2.8 double filter_t::alpha2

rescaled f2

Definition at line 86 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and create_filter().

6.10.2.9 double filter_t::w_alpha1

warped alpha1

Definition at line 88 of file bpm_dsp.h.

Referenced by create_filter(), and normalise_filter().

6.10.2.10 double filter_t::w_alpha2

warped alpha2

Definition at line 89 of file bpm_dsp.h.

Referenced by create_filter(), and normalise_filter().

6.10.2.11 double filter_t::cheb_ripple

ripple for chebyshev filters

Definition at line 91 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

6.10.2.12 double filter_t::Q

Q factor for resonators

Definition at line 92 of file bpm_dsp.h.

Referenced by create_filter(), and create_resonator_representation().

6.10.2.13 complex_t filter_t::dc_gain

Complex DC gain of the filter

Definition at line 94 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and print_filter().

6.10.2.14 complex_t filter_t::fc_gain

Complex Center frequency gain of filter

Definition at line 95 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and print_filter().

6.10.2.15 complex_t filter_t::hf_gain

Complex High frequency (fNy) gain of filter

Definition at line 96 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and print_filter().

6.10.2.16 double filter_t::gain

Actual Filter gain

Definition at line 97 of file bpm_dsp.h.

Referenced by apply_filter(), and calculate_filter_coefficients().

6.10.2.17 filterrep_t* filter_t::cplane

pointer to complex filter representation, poles and zeros

Definition at line 99 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), create_filter(), delete_filter(), and print_filter().

6.10.2.18 int filter_t::IsFIR

filter is FIR

Definition at line 101 of file bpm_dsp.h.

Referenced by apply_filter(), and create_filter().

6.10.2.19 int filter_t::nxc

number of x coefficients

Definition at line 102 of file bpm_dsp.h.

Referenced by apply_filter(), calculate_filter_coefficients(), and print_filter().

6.10.2.20 double filter_t::xc[MAXPZ+1]

pointer to array of x coefficients

Definition at line 103 of file bpm_dsp.h.

Referenced by apply_filter(), calculate_filter_coefficients(), and print_filter().

6.10.2.21 int filter_t::nyc

number of y coefficients (for IIR filters)

Definition at line 104 of file bpm_dsp.h.

Referenced by apply_filter(), and calculate_filter_coefficients().

6.10.2.22 double filter_t::yc[MAXPZ+1]

pointer to array of y coefficients

Definition at line 105 of file bpm_dsp.h.

Referenced by apply_filter(), calculate_filter_coefficients(), and create_filter().

6.10.2.23 double filter_t::xv[MAXPZ+1]

filter x buffer, used in apply_filter

Definition at line 107 of file bpm_dsp.h.

Referenced by apply_filter().

6.10.2.24 double filter_t::yv[MAXPZ+1]

filter y buffer, used in apply_filter

Definition at line 108 of file bpm_dsp.h.

Referenced by apply_filter().

6.10.2.25 int filter_t::ns

number of samples of waveforms to be filtered

Definition at line 110 of file bpm_dsp.h.

Referenced by apply_filter(), create_filter(), filter_impulse_response(), and filter_step_response().

6.10.2.26 double* filter_t::wfbuffer

waveform buffer for filter computations, allocated once !

Definition at line 111 of file bpm_dsp.h.

Referenced by apply_filter(), create_filter(), and delete_filter().

The documentation for this struct was generated from the following file:

- bpmdsp/**bpm_dsp.h**

6.11 filterrep_t Struct Reference

`#include <bpm_dsp.h>`

Collaboration diagram for filterrep_t:

6.11.1 Detailed Description

The filter representation in the complex plane (poles/zeros).

Definition at line 65 of file bpm_dsp.h.

Data Fields

- int **npoles**
- int **nzeros**
- **complex_t pole** [MAXPZ]
- **complex_t zero** [MAXPZ]

6.11.2 Field Documentation

6.11.2.1 int filterrep_t::npoles

The number of filter poles

Definition at line 66 of file bpm_dsp.h.

Referenced by _add_splane_pole(), calculate_filter_coefficients(), create_filter(), create_resonator_representation(), create_splane_representation(), normalise_filter(), print_filter_representation(), and zplane_transform().

6.11.2.2 int filterrep_t::nzeros

The number of filter zeros

Definition at line 67 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), create_resonator_representation(), normalise_filter(), print_filter_representation(), and zplane_transform().

6.11.2.3 complex_t filterrep_t::pole[MAXPZ]

Array of the filter's complex poles

Definition at line 68 of file bpm_dsp.h.

Referenced by _add_splane_pole(), calculate_filter_coefficients(), create_resonator_representation(), normalise_filter(), print_filter_representation(), and zplane_transform().

6.11.2.4 complex_t filterrep_t::zero[MAXPZ]

Array of the filter's complex zeros

Definition at line 69 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), create_resonator_representation(), normalise_filter(), print_filter_representation(), and zplane_transform().

The documentation for this struct was generated from the following file:

- bpmdsp/**bpm_dsp.h**

6.12 gsl_block_struct Struct Reference

6.12.1 Detailed Description

Definition at line 146 of file bpm_nr.h.

Data Fields

- size_t **size**
- double * **data**

The documentation for this struct was generated from the following file:

- bpmnr/**bpm_nr.h**

6.13 gsl_matrix Struct Reference

Collaboration diagram for gsl_matrix:

6.13.1 Detailed Description

Definition at line 156 of file bpm_nr.h.

Data Fields

- size_t **size1**
- size_t **size2**
- size_t **tda**
- double * **data**
- **gsl_block** * **block**
- int **owner**

The documentation for this struct was generated from the following file:

- bpmnr/**bpm_nr.h**

6.14 gsl_vector Struct Reference

Collaboration diagram for gsl_vector:

6.14.1 Detailed Description

Definition at line 176 of file bpm_nr.h.

Data Fields

- size_t **size**
- size_t **stride**
- double * **data**
- **gsl_block** * **block**
- int **owner**

The documentation for this struct was generated from the following file:

- bpmnr/**bpm_nr.h**

6.15 lm_fstate Struct Reference

`#include <bpm_nr.h>`

6.15.1 Detailed Description

structure needed for levenberg marquard minimisation

Definition at line 118 of file bpm_nr.h.

Data Fields

- int **n**
- int * **nfev**
- double * **hx**
- double * **x**
- void * **adata**

The documentation for this struct was generated from the following file:

- bpmnr/**bpm_nr.h**

6.16 m33 Struct Reference

`#include <bpm_orbit.h>`

6.16.1 Detailed Description

Structure representing a 3x3-matrix, for use in the orbit generation routines

Definition at line 49 of file bpm_orbit.h.

Data Fields

- double **e** [3][3]

6.16.2 Field Documentation

6.16.2.1 double m33::e[3][3]

the matrix

Definition at line 50 of file bpm_orbit.h.

Referenced by m_matadd(), m_matmult(), m_print(), m_rotmat(), and v_matmult().

The documentation for this struct was generated from the following file:

- bpmorbit/**bpm_orbit.h**

6.17 v3 Struct Reference

`#include <bpm_orbit.h>`

6.17.1 Detailed Description

Structure representing a 3-vector, for use in the orbit generation routines

Definition at line 38 of file bpm_orbit.h.

Data Fields

- double **x**
- double **y**
- double **z**

6.17.2 Field Documentation

6.17.2.1 double v3::x

x-coordinate

Definition at line 39 of file bpm_orbit.h.

Referenced by generate_bpm_orbit(), v_add(), v_copy(), v_cross(), v_dot(), v_matmult(), v_print(), v_scale(), and v_sub().

6.17.2.2 double v3::y

y-coordinate

Definition at line 40 of file bpm_orbit.h.

Referenced by generate_bpm_orbit(), v_add(), v_copy(), v_cross(), v_dot(), v_matmult(), v_print(), v_scale(), and v_sub().

6.17.2.3 double v3::z

z-coordinate

Definition at line 41 of file bpm_orbit.h.

Referenced by generate_bpm_orbit(), v_add(), v_copy(), v_cross(), v_dot(), v_matmult(), v_print(), v_scale(), and v_sub().

The documentation for this struct was generated from the following file:

- bpmorbit/**bpm_orbit.h**

7 libbpm File Documentation

7.1 bpm_units.h File Reference

7.1.1 Detailed Description

Physical unit definitions for libbpm.

Definition in file **bpm_units.h**.

Defines

- #define **Hz**
- #define **kHz**
- #define **MHz**
- #define **GHz**
- #define **sec**
- #define **msec**
- #define **usec**
- #define **nsec**
- #define **eV**
- #define **keV**
- #define **MeV**
- #define **GeV**
- #define **rad**
- #define **mrad**
- #define **urad**
- #define **nrad**
- #define **mC**
- #define **uC**
- #define **nC**
- #define **pC**
- #define **meter**
- #define **mmeter**
- #define **umeter**
- #define **nmeter**
- #define **Volt**
- #define **mVolt**
- #define **nVolt**

7.2 bpmalloc/alloc_complex_wave_double.c File Reference

7.2.1 Detailed Description

Definition in file **alloc_complex_wave_double.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_alloc.h>
```

Include dependency graph for alloc_complex_wave_double.c:

Functions

- double ** **alloc_complex_wave_double** (int ns)
- void **free_complex_wave_double** (double **w, int ns)

7.3 bpmalloc/alloc_simple_wave_double.c File Reference

7.3.1 Detailed Description

Definition in file **alloc_simple_wave_double.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_alloc.h>
```

Include dependency graph for alloc_simple_wave_double.c:

Functions

- double * **alloc_simple_wave_double** (int ns)
- void **free_simple_wave_double** (double *w)

7.4 bpmalloc/alloc_simple_wave_int.c File Reference

7.4.1 Detailed Description

Definition in file **alloc_simple_wave_int.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_alloc.h>
```

Include dependency graph for alloc_simple_wave_int.c:

Functions

- int * **alloc_simple_wave_int** (int ns)
- void **free_simple_wave_int** (int *w)

7.5 bpmalloc/bpm_alloc.h File Reference

7.5.1 Detailed Description

libbpm waveform memory allocation routines

This header contains the definitions for the memory allocation routines to handle waveforms in libbpm.

Definition in file **bpm_alloc.h**.

```
#include <stdlib.h>
```

```
#include <bpm/bpm_defs.h>
```

Include dependency graph for bpm_alloc.h:

Functions

- EXTERN double ** **alloc_complex_wave_double** (int ns)
- EXTERN void **free_complex_wave_double** (double **w, int ns)
- EXTERN double * **alloc_simple_wave_double** (int ns)
- EXTERN void **free_simple_wave_double** (double *w)
- EXTERN int * **alloc_simple_wave_int** (int ns)
- EXTERN void **free_simple_wave_int** (int *w)

7.6 bpmanalysis/ana_compute_residual.c File Reference

7.6.1 Detailed Description

Definition in file **ana_compute_residual.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_analysis.h>
```

Include dependency graph for ana_compute_residual.c:

Functions

- int **ana_compute_residual** (**bpmproc_t** **proc, int num_bpms, int num_evts, double *coeffs, int mode, double *mean, double *rms)

7.7 bpmanalysis/ana_def_cutfn.c File Reference

7.7.1 Detailed Description

Definition in file **ana_def_cutfn.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_analysis.h>
```

Include dependency graph for ana_def_cutfn.c:

Functions

- int **ana_def_cutfn** (**bpmproc_t** *proc)

Variables

- int(*) **ana_cutfn** (**bpmproc_t** *proc)

7.8 bpmanalysis/ana_get_svd_coeffs.c File Reference

7.8.1 Detailed Description

Definition in file **ana_get_svd_coeffs.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_analysis.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for ana_get_svd_coeffs.c:

Functions

- int **ana_get_svd_coeffs** (**bpmproc_t** **proc, int num_bpms, int num_svd, int total_num_evts, double *coeffs, int mode)

7.9 bpmanalysis/ana_set_cutfn.c File Reference

7.9.1 Detailed Description

Definition in file **ana_set_cutfn.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_analysis.h>
```

Include dependency graph for ana_set_cutfn.c:

Functions

- int **ana_set_cutfn** (int(*cutfn)(**bpmproc_t** *proc))

7.10 bpmanalysis/bpm_analysis.h File Reference

7.10.1 Detailed Description

libbpm analysis routines

This header contains definitions for the libbpm BPM data analysis routines. These mainly are the SVD and resolution/residual calculation routines along with the definition of an analysis cut function...

Definition in file **bpm_analysis.h**.

```
#include <math.h>
#include <bpm/bpm_defs.h>
#include <bpm/bpm_interface.h>
```

Include dependency graph for bpm_analysis.h:

Defines

- #define **BPM_GOOD_EVENT**
- #define **BPM_BAD_EVENT**
- #define **ANA_SVD_TILT**
- #define **ANA_SVD_NOTILT**

Functions

- EXTERN int **ana_set_cutfn** (int(*cutfn)(**bpmproc_t** *proc))
- EXTERN int **ana_get_svd_coeffs** (**bpmproc_t** **proc, int num_bpms, int num_svd, int total_num_evts, double *coeffs, int mode)
- EXTERN int **ana_compute_residual** (**bpmproc_t** **proc, int num_bpms, int num_evts, double *coeffs, int mode, double *mean, double *rms)
- EXTERN int **ana_def_cutfn** (**bpmproc_t** *proc)

Variables

- EXTERN int(*) **ana_cutfn** (**bpmproc_t** *proc)

7.11 bpmcalibration/bpm_calibration.h File Reference

7.11.1 Detailed Description

calibration routines

This header contains some BPM calibration routines

Definition in file **bpm_calibration.h**.

```
#include <math.h>
```

```
#include <bpm/bpm_defs.h>
```

```
#include <bpm/bpm_interface.h>
```

Include dependency graph for bpm_calibration.h:

Functions

- EXTERN int **setup_calibration** (**bpmconf_t** *cnf, **bpmproc_t** *proc, int npulses, int startpulse, int stoppulse, double angle, double startpos, double endpos, int num_steps, **beamconf_t** *beam)
- EXTERN int **calibrate** (**bpmconf_t** *bpm, **beamconf_t** *beam, **bpmproc_t** *proc, int npulses, **bpmcalib_t** *cal)
- EXTERN int **update_freq_tdecay** (**bpmproc_t** *proc, int npulses, **bpmcalib_t** *cal)
- EXTERN int **calibrate_svd** (**beamconf_t** **beam, **bpmconf_t** **bpm, **bpmproc_t** **proc, int npulses, int nbpms, int *bpmidx, **bpmcalib_t** *cal)

- EXTERN int **save_calibration** (char *fname, **bpmconf_t** *bpm, **bpmcalib_t** *cal, int num_bpms)
- EXTERN int **load_calibration** (char *fname, **bpmconf_t** *bpm, **bpmcalib_t** *cal, int num_bpms)

7.12 bpmcalibration/calibrate.c File Reference

7.12.1 Detailed Description

Definition in file **calibrate.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_calibration.h>
```

```
#include <bpm/bpm_nr.h>
```

Include dependency graph for calibrate.c:

Functions

- int **calibrate** (**bpmconf_t** *bpm, **beamconf_t** *beam, **bpmproc_t** *proc, int npulses, **bpmcalib_t** *cal)

7.13 bpmcalibration/calibrate_simple.c File Reference

7.13.1 Detailed Description

Definition in file **calibrate_simple.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_calibration.h>
```

Include dependency graph for calibrate_simple.c:

Functions

- int **calibrate_simple** (**bpmconf_t** **bpmcnf, **bpmproc_t** **proc, **beamconf_t** **beam, int npulses)

7.13.2 Function Documentation

7.13.2.1 int calibrate_simple (bpmconf_t ** *bpmcnf*, bpmproc_t ** *proc*, beamconf_t ** *beam*, int *npulses*)

Definition at line 7 of file calibrate_simple.c.

References bpm_error().

7.14 bpmcalibration/calibrate_svd.c File Reference

7.14.1 Detailed Description

Definition in file **calibrate_svd.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_calibration.h>
```

Include dependency graph for calibrate_svd.c:

Functions

- int **calibrate_svd** (**beamconf_t** **beam, **bpmconf_t** **cnf, **bpmproc_t** **proc, int npulses, int nbpms, int *bpmidx, **bpmcalib_t** *cal)

7.15 bpmcalibration/load_calibration.c File Reference

7.15.1 Detailed Description

Definition in file **load_calibration.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_calibration.h>
```

Include dependency graph for load_calibration.c:

Functions

- int **load_calibration** (char *fname, **bpmconf_t** *bpm, **bpmcalib_t** *cal, int num_bpms)

7.16 bpmcalibration/save_calibration.c File Reference

7.16.1 Detailed Description

Definition in file **save_calibration.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_calibration.h>
```

Include dependency graph for save_calibration.c:

Functions

- int **save_calibration** (char *fname, **bpmconf_t** *bpm, **bpmcalib_t** *cal, int num_bpms)

7.17 bpmcalibration/setup_calibration.c File Reference

7.17.1 Detailed Description

Definition in file **setup_calibration.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_calibration.h>
```

Include dependency graph for setup_calibration.c:

Functions

- int **setup_calibration** (**bpmconf_t** *cnf, **bpmproc_t** *proc, int npulses, int startpulse, int stoppulse, double angle, double startpos, double endpos, int num_steps, **beamconf_t** *beam)

7.18 bpmcalibration/update_freq_tdecay.c File Reference

7.18.1 Detailed Description

Definition in file **update_freq_tdecay.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_calibration.h>
```

Include dependency graph for update_freq_tdecay.c:

Functions

- int **update_freq_tdecay** (**bpmproc_t** *proc, int npulses, **bpmcalib_t** *cal)

7.19 bpmdsp/bpm_dsp.h File Reference

7.19.1 Detailed Description

libbpm digital signal processing routines

This header contains the definitions for the digital signal processing routines for libbpm.

Definition in file **bpm_dsp.h**.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bpm/bpm_defs.h"
```

```
#include "bpm/bpm_messages.h"
#include "bpm/bpm_alloc.h"
#include "bpm/bpm_nr.h"
```

Include dependency graph for bpm_dsp.h:

Data Structures

- struct **filterrep_t**
- struct **filter_t**

Defines

- #define **BESSEL**
- #define **BUTTERWORTH**
- #define **CHEBYSHEV**
- #define **RAISEDCOSINE**
- #define **RESONATOR**
- #define **GAUSSIAN**
- #define **BILINEAR_Z_TRANSFORM**
- #define **MATCHED_Z_TRANSFORM**
- #define **NO_PREWARP**
- #define **LOWPASS**
- #define **HIGHPASS**
- #define **BANDPASS**
- #define **BANDSTOP**
- #define **NOTCH**
- #define **ALLPASS**
- #define **MAXORDER**
- #define **MAXPZ**
- #define **FILT_EPS**
- #define **MAX_RESONATOR_ITER**

Functions

- EXTERN **filter_t** * **create_filter** (char name[], unsigned int options, int order, int ns, double fs, double f1, double f2, double par)
- EXTERN int **apply_filter** (**filter_t** *f, double *wf)
- EXTERN void **print_filter** (FILE *of, **filter_t** *f)
- EXTERN void **delete_filter** (**filter_t** *f)
- EXTERN int **filter_step_response** (**filter_t** *f, double *wf, int itrig)
- EXTERN int **filter_impulse_response** (**filter_t** *f, double *wf, int itrig)
- EXTERN **filterrep_t** * **create_splane_representation** (**filter_t** *f)
- EXTERN **filterrep_t** * **create_resonator_representation** (**filter_t** *f)
- EXTERN **filterrep_t** * **zplane_transform** (**filter_t** *f, **filterrep_t** *s)
- EXTERN void **print_filter_representation** (FILE *of, **filterrep_t** *r)
- EXTERN int **normalise_filter** (**filter_t** *f, **filterrep_t** *s)

- EXTERN int **calculate_filter_coefficients** (**filter_t** *f)
- EXTERN int **_expand_complex_polynomial** (**complex_t** *w, int n, **complex_t** *a)
- EXTERN **complex_t** **_eval_complex_polynomial** (**complex_t** *a, int n, **complex_t** z)

7.20 bpmdsp/calculate_filter_coefficients.c File Reference

7.20.1 Detailed Description

Definition in file **calculate_filter_coefficients.c**.

```
#include "bpm/bpm_dsp.h"
```

Include dependency graph for calculate_filter_coefficients.c:

Functions

- int **_expand_complex_polynomial** (**complex_t** *w, int n, **complex_t** *a)
- **complex_t** **_eval_complex_polynomial** (**complex_t** *a, int n, **complex_t** z)
- int **calculate_filter_coefficients** (**filter_t** *f)

7.21 bpmdsp/create_filter.c File Reference

7.21.1 Detailed Description

Definition in file **create_filter.c**.

```
#include <string.h>
#include "bpm/bpm_alloc.h"
#include "bpm/bpm_dsp.h"
```

Include dependency graph for create_filter.c:

Functions

- **filter_t** * **create_filter** (char name[], unsigned int options, int order, int ns, double fs, double f1, double f2, double par)

7.22 bpmdsp/create_resonator_representation.c File Reference

7.22.1 Detailed Description

Definition in file **create_resonator_representation.c**.

```
#include "bpm/bpm_dsp.h"
```

Include dependency graph for create_resonator_representation.c:

Functions

- **complex_t _reflect** (**complex_t** z)
- **filterrep_t** * **create_resonator_representation** (**filter_t** *f)

7.23 bpmdsp/create_splane_representation.c File Reference

7.23.1 Detailed Description

Definition in file **create_splane_representation.c**.

```
#include "bpm/bpm_dsp.h"
```

Include dependency graph for create_splane_representation.c:

Functions

- void **_add_splane_pole** (**filterrep_t** *r, **complex_t** z)
- **filterrep_t** * **create_splane_representation** (**filter_t** *f)

7.24 bpmdsp/delete_filter.c File Reference

7.24.1 Detailed Description

Definition in file **delete_filter.c**.

```
#include "bpm/bpm_dsp.h"
#include "bpm/bpm_alloc.h"
```

Include dependency graph for delete_filter.c:

Functions

- void **delete_filter** (**filter_t** *f)

7.25 bpmdsp/filter_impulse_response.c File Reference

7.25.1 Detailed Description

Definition in file **filter_impulse_response.c**.

```
#include "bpm/bpm_dsp.h"
```

Include dependency graph for filter_impulse_response.c:

Functions

- int **filter_impulse_response** (**filter_t** *f, double *wf, int itrig)

7.26 bpmdsp/filter_step_response.c File Reference

7.26.1 Detailed Description

Definition in file **filter_step_response.c**.

```
#include "bpm/bpm_dsp.h"
```

Include dependency graph for filter_step_response.c:

Functions

- int **filter_step_response** (**filter_t** *f, double *wf, int itrig)

7.27 bpmdsp/normalise_filter.c File Reference

7.27.1 Detailed Description

Definition in file **normalise_filter.c**.

```
#include "bpm/bpm_dsp.h"
```

Include dependency graph for normalise_filter.c:

Functions

- int **normalise_filter** (**filter_t** *f, **filterrep_t** *s)

7.28 bpmdsp/print_filter.c File Reference

7.28.1 Detailed Description

Definition in file **print_filter.c**.

```
#include "bpm/bpm_dsp.h"
```

Include dependency graph for print_filter.c:

Functions

- void **print_filter** (FILE *of, **filter_t** *f)

7.29 bpmdsp/print_filter_representation.c File Reference

7.29.1 Detailed Description

Definition in file **print_filter_representation.c**.

```
#include "bpm/bpm_dsp.h"
```

Include dependency graph for print_filter_representation.c:

Functions

- void **print_filter_representation** (FILE *of, **filterrep_t** *r)

7.30 bpmdsp/zplane_transform.c File Reference

7.30.1 Detailed Description

Definition in file **zplane_transform.c**.

```
#include "bpm/bpm_dsp.h"
```

Include dependency graph for zplane_transform.c:

Functions

- **filterrep_t** * **zplane_transform** (**filter_t** *f, **filterrep_t** *s)

7.31 bpminterface/bpm_interface.h File Reference

7.31.1 Detailed Description

Front end interface structure definitions and handlers.

This header contains the front-end interface structures and handlers for libbpm. They define a set of user friendly structures like bpmconf_t, bpmcalib_t, beamconf_t etc... to work with the bpm data.

Definition in file **bpm_interface.h**.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <bpm/bpm_defs.h>
```

Include dependency graph for bpm_interface.h:

Data Structures

- struct **bpmconf**
- struct **bpmsignal**
- struct **bpmcalib**
- struct **bpmproc**
- struct **beamconf**

Typedefs

- typedef **bpmconf bpmconf_t**
- typedef **bpmsignal bpmsignal_t**
- typedef **bpmcalib bpmcalib_t**
- typedef **bpmproc bpmproc_t**
- typedef **beamconf beamconf_t**

Enumerations

- enum **bpmtype_t** { **diode**, **monopole**, **dipole** }
- enum **bpmpol_t** { **horiz**, **vert** }
- enum **bpmphase_t** { **randomised**, **locked** }
- enum **rffiltertype_t** { **nofilter**, **butterworth_low_pass**, **butterworth_band_pass**, **butterworth_high_pass** }

Functions

- EXTERN int **load_bpmconf** (const char *fname, **bpmconf_t** **conf, int *num_conf)
- EXTERN int **get_header** (FILE *file, double *version, int *num_structs)
- EXTERN int **load_struct** (FILE *file, char ***arg_list, char ***val_list, int *num_args)
- EXTERN int **save_signals** (char *fname, **bpmsignal_t** *sigs, int num_evts)
- EXTERN int **load_signals** (char *fname, **bpmsignal_t** **sigs)

Variables

- EXTERN int **bpm_verbose**

7.32 bpminterface/get_header.c File Reference

7.32.1 Detailed Description

Definition in file **get_header.c**.

```
#include <bpm/bpm_interface.h>
```

```
#include <bpm/bpm_messages.h>
```

Include dependency graph for get_header.c:

Functions

- int **get_header** (FILE *file, double *version, int *num_structs)

7.33 bpminterface/load_bpmconf.c File Reference

7.33.1 Detailed Description

Definition in file **load_bpmconf.c**.

```
#include <stdio.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_interface.h>
#include <bpm/bpm_version.h>
#include <bpm/bpm_units.h>
```

Include dependency graph for load_bpmconf.c:

Functions

- int **load_bpmconf** (const char *fname, **bpmconf_t** **conf, int *num_conf)

7.34 bpminterface/load_signals.c File Reference

7.34.1 Detailed Description

Definition in file **load_signals.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_version.h>
```

Include dependency graph for load_signals.c:

Functions

- int **load_signals** (char *fname, **bpmsignal_t** **sigs)

7.35 bpminterface/load_struct.c File Reference

7.35.1 Detailed Description

Definition in file **load_struct.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_messages.h>
```

Include dependency graph for load_struct.c:

Defines

- #define **MAX_ARGS**

Functions

- int **load_struct** (FILE *file, char ***arg_list, char ***val_list, int *num_args)

7.36 bpminterface/save_signals.c File Reference

7.36.1 Detailed Description

Definition in file **save_signals.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_version.h>
```

Include dependency graph for save_signals.c:

Functions

- int **save_signals** (char *fname, **bpmsignal_t** *sigs, int num_evts)

7.37 bpmmessages/bpm_error.c File Reference

7.37.1 Detailed Description

Definition in file **bpm_error.c**.

```
#include <stdio.h>
#include <bpm/bpm_messages.h>
```

Include dependency graph for bpm_error.c:

Functions

- void **bpm_error** (char *msg, char *f, int l)

7.38 bpmmessages/bpm_messages.h File Reference

7.38.1 Detailed Description

libbpm error/warning messages

This header defines the routines which take care of printing error and warning messages

Definition in file **bpm_messages.h**.

`#include <bpm/bpm_defs.h>`

Include dependency graph for bpm_messages.h:

Functions

- EXTERN void **bpm_error** (char *msg, char *f, int l)
- EXTERN void **bpm_warning** (char *msg, char *f, int l)

7.39 bpmmessages/bpm_warning.c File Reference

7.39.1 Detailed Description

Definition in file **bpm_warning.c**.

`#include <stdio.h>`

`#include <bpm/bpm_messages.h>`

Include dependency graph for bpm_warning.c:

Functions

- void **bpm_warning** (char *msg, char *f, int l)

7.40 bpmnr/bpm_nr.h File Reference

7.40.1 Detailed Description

libbpm numerical helper routines

Header file containing the numerical recipies and GNU Scientific Library routines used in the library.

Definition in file **bpm_nr.h**.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <math.h>`

`#include <float.h>`

```
#include <string.h>
#include <bpm/bpm_defs.h>
```

Include dependency graph for bpm_nr.h:

Data Structures

- struct **lm_fstate**
- struct **gsl_block_struct**
- struct **gsl_matrix**
- struct **_gsl_matrix_view**
- struct **gsl_vector**
- struct **_gsl_vector_view**
- struct **_gsl_vector_const_view**
- struct **complex_t**

Defines

- #define **GCF_ITMAX**
- #define **GCF_FPMIN**
- #define **GCF_EPS**
- #define **GSER_EPS**
- #define **GSER_ITMAX**
- #define **RAN1_IA**
- #define **RAN1_IM**
- #define **RAN1_AM**
- #define **RAN1_IQ**
- #define **RAN1_IR**
- #define **RAN1_NTAB**
- #define **RAN1_NDIV**
- #define **RAN1_EPS**
- #define **RAN1_RNMX**
- #define **__LM_BLOCKSZ__**
- #define **__LM_BLOCKSZ__SQ**
- #define **LINSOLVERS_RETAIN_MEMORY**
- #define **__LM_STATIC__**
- #define **FABS**(x)
- #define **CNST**(x)
- #define **_LM_POW_**
- #define **LM_DER_WORKSZ**(npar, nmeas)
- #define **LM_DIF_WORKSZ**(npar, nmeas)
- #define **LM_EPSILON**
- #define **LM_ONE_THIRD**
- #define **LM_OPTS_SZ**
- #define **LM_INFO_SZ**
- #define **LM_INIT_MU**
- #define **LM_STOP_THRESH**
- #define **LM_DIFF_DELTA**

- #define **NR_FFTFORWARD**
- #define **NR_FFTBACKWARD**
- #define **__LM_MEDIAN3**(a, b, c)
- #define **NULL_VECTOR**
- #define **NULL_VECTOR_VIEW**
- #define **NULL_MATRIX**
- #define **NULL_MATRIX_VIEW**
- #define **GSL_DBL_EPSILON**
- #define **OFFSET**(N, incX)
- #define **GSL_MIN**(a, b)

Typedefs

- typedef enum **CBLAS_TRANSPOSE CBLAS_TRANSPOSE_t**
- typedef **gsl_block_struct gsl_block**
- typedef **_gsl_matrix_view gsl_matrix_view**
- typedef **_gsl_vector_view gsl_vector_view**
- typedef const **_gsl_vector_const_view gsl_vector_const_view**

Enumerations

- enum **CBLAS_TRANSPOSE** { **CblasNoTrans**, **CblasTrans**, **CblasConjTrans** }
- enum **CBLAS_ORDER** { **CblasRowMajor**, **CblasColMajor** }

Functions

- EXTERN double **nr_gammln** (double xx)
- EXTERN double **nr_gammq** (double a, double x)
- EXTERN int **nr_gcf** (double *gammcf, double a, double x, double *gln)
- EXTERN int **nr_gser** (double *gamser, double a, double x, double *gln)
- EXTERN int **nr_fit** (double *x, double y[], int ndata, double sig[], int mwt, double *a, double *b, double *siga, double *sigb, double *chi2, double *q)
- EXTERN int **nr_is_pow2** (unsigned long n)
- EXTERN int **nr_four1** (double data[], unsigned long nn, int isign)
- EXTERN int **nr_realft** (double data[], unsigned long n, int isign)
- EXTERN double **nr_ran1** (long *idum)
- EXTERN int **nr_seed** (long seed)
- EXTERN double **nr_ranuniform** (double lower, double upper)
- EXTERN double **nr_rangauss** (double mean, double std_dev)
- EXTERN int **nr_lmder** (void(*func)(double *p, double *hx, int m, int n, void *adata), void(*jacf)(double *p, double *j, int m, int n, void *adata), double *p, double *x, int m, int n, int itmax, double *opts, double *info, double *work, double *covar, void *adata)
- EXTERN int **nr_lmdif** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *x, int m, int n, int itmax, double *opts, double *info, double *work, double *covar, void *adata)
- EXTERN int **nr_lmder_bc** (void(*func)(double *p, double *hx, int m, int n, void *adata), void(*jacf)(double *p, double *j, int m, int n, void *adata), double *p, double *x, int m, int n, double *lb, double *ub, int itmax, double *opts, double *info, double *work, double *covar, void *adata)

- EXTERN int **nr_lmdif_bc** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *x, int m, int n, double *lb, double *ub, int itmax, double *opts, double *info, double *work, double *covar, void *adata)
- EXTERN void **nr_lmchkjac** (void(*func)(double *p, double *hx, int m, int n, void *adata), void(*jacf)(double *p, double *j, int m, int n, void *adata), double *p, int m, int n, void *adata, double *err)
- EXTERN int **nr_lmcovar** (double *JtJ, double *C, double sumsq, int m, int n)
- EXTERN int **nr_ax_eq_b_LU** (double *A, double *B, double *x, int n)
- EXTERN void **nr_trans_mat_mat_mult** (double *a, double *b, int n, int m)
- EXTERN void **nr_fdif_forw_jac_approx** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *hx, double *hxx, double delta, double *jac, int m, int n, void *adata)
- EXTERN void **nr_fdif_cent_jac_approx** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *hxm, double *hxp, double delta, double *jac, int m, int n, void *adata)
- EXTERN double **nr_median** (int n, double *arr)
- EXTERN double **nr_select** (int k, int n, double *org_arr)
- EXTERN **gsl_matrix** * **gsl_matrix_calloc** (const size_t n1, const size_t n2)
- EXTERN **_gsl_vector_view gsl_matrix_column** (**gsl_matrix** *m, const size_t i)
- EXTERN **_gsl_matrix_view gsl_matrix_submatrix** (**gsl_matrix** *m, const size_t i, const size_t j, const size_t n1, const size_t n2)
- EXTERN double **gsl_matrix_get** (const **gsl_matrix** *m, const size_t i, const size_t j)
- EXTERN void **gsl_matrix_set** (**gsl_matrix** *m, const size_t i, const size_t j, const double x)
- EXTERN int **gsl_matrix_swap_columns** (**gsl_matrix** *m, const size_t i, const size_t j)
- EXTERN **gsl_matrix** * **gsl_matrix_alloc** (const size_t n1, const size_t n2)
- EXTERN **_gsl_vector_const_view gsl_matrix_const_row** (const **gsl_matrix** *m, const size_t i)
- EXTERN **_gsl_vector_view gsl_matrix_row** (**gsl_matrix** *m, const size_t i)
- EXTERN **_gsl_vector_const_view gsl_matrix_const_column** (const **gsl_matrix** *m, const size_t j)
- EXTERN void **gsl_matrix_set_identity** (**gsl_matrix** *m)
- EXTERN **gsl_vector** * **gsl_vector_calloc** (const size_t n)
- EXTERN **_gsl_vector_view gsl_vector_subvector** (**gsl_vector** *v, size_t offset, size_t n)
- EXTERN double **gsl_vector_get** (const **gsl_vector** *v, const size_t i)
- EXTERN void **gsl_vector_set** (**gsl_vector** *v, const size_t i, double x)
- EXTERN int **gsl_vector_swap_elements** (**gsl_vector** *v, const size_t i, const size_t j)
- EXTERN **_gsl_vector_const_view gsl_vector_const_subvector** (const **gsl_vector** *v, size_t i, size_t n)
- EXTERN void **gsl_vector_free** (**gsl_vector** *v)
- EXTERN int **gsl_linalg_SV_solve** (const **gsl_matrix** *U, const **gsl_matrix** *Q, const **gsl_vector** *S, const **gsl_vector** *b, **gsl_vector** *x)
- EXTERN int **gsl_linalg_bidiag_unpack** (const **gsl_matrix** *A, const **gsl_vector** *tau_U, **gsl_matrix** *U, const **gsl_vector** *tau_V, **gsl_matrix** *V, **gsl_vector** *diag, **gsl_vector** *superdiag)
- EXTERN int **gsl_linalg_householder_hm** (double tau, const **gsl_vector** *v, **gsl_matrix** *A)
- EXTERN int **gsl_linalg_bidiag_unpack2** (**gsl_matrix** *A, **gsl_vector** *tau_U, **gsl_vector** *tau_V, **gsl_matrix** *V)

- EXTERN int **gsl_linalg_householder_hm1** (double tau, **gsl_matrix** *A)
- EXTERN void **create_givens** (const double a, const double b, double *c, double *s)
- EXTERN double **gsl_linalg_householder_transform** (**gsl_vector** *v)
- EXTERN int **gsl_linalg_householder_mh** (double tau, const **gsl_vector** *v, **gsl_matrix** *A)
- EXTERN void **chop_small_elements** (**gsl_vector** *d, **gsl_vector** *f)
- EXTERN void **qrstep** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, **gsl_matrix** *V)
- EXTERN double **trailing_eigenvalue** (const **gsl_vector** *d, const **gsl_vector** *f)
- EXTERN void **create_schur** (double d0, double f0, double d1, double *c, double *s)
- EXTERN void **svd2** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, **gsl_matrix** *V)
- EXTERN void **chase_out_intermediate_zero** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, size_t k0)
- EXTERN void **chase_out_trailing_zero** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *V)
- EXTERN int **gsl_isnan** (const double x)
- EXTERN double **gsl_blas_dnrm2** (const **gsl_vector** *X)
- EXTERN double **cblas_dnrm2** (const int N, const double *X, const int incX)
- EXTERN void **gsl_blas_dscal** (double alpha, **gsl_vector** *X)
- EXTERN void **cblas_dscal** (const int N, const double alpha, double *X, const int incX)
- EXTERN void **cblas_dgemv** (const enum **CBLAS_ORDER** order, const enum **CBLAS_TRANSPOSE** TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)
- EXTERN **gsl_block** * **gsl_block_alloc** (const size_t n)
- EXTERN void **gsl_block_free** (**gsl_block** *b)
- EXTERN **complex_t** **complex** (double re, double im)
- EXTERN double **c_real** (**complex_t** z)
- EXTERN double **c_imag** (**complex_t** z)
- EXTERN **complex_t** **c_conj** (**complex_t** z)
- EXTERN **complex_t** **c_neg** (**complex_t** z)
- EXTERN **complex_t** **c_sum** (**complex_t** z1, **complex_t** z2)
- EXTERN **complex_t** **c_diff** (**complex_t** z1, **complex_t** z2)
- EXTERN **complex_t** **c_mult** (**complex_t** z1, **complex_t** z2)
- EXTERN **complex_t** **c_div** (**complex_t** z1, **complex_t** z2)
- EXTERN **complex_t** **c_scale** (double r, **complex_t** z)
- EXTERN **complex_t** **c_sqr** (**complex_t** z)
- EXTERN **complex_t** **c_sqrt** (**complex_t** z)
- EXTERN double **c_norm2** (**complex_t** z)
- EXTERN double **c_abs** (**complex_t** z)
- EXTERN double **c_arg** (**complex_t** z)
- EXTERN **complex_t** **c_exp** (**complex_t** z)
- EXTERN int **c_isequal** (**complex_t** z1, **complex_t** z2)

Variables

- EXTERN long **bpm_rseed**

7.41 bpmnr/gsl_blas.c File Reference

7.41.1 Detailed Description

Definition in file **gsl_blas.c**.

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_nr.h>`

Include dependency graph for gsl_blas.c:

Functions

- double **gsl_blas_dnrm2** (const **gsl_vector** *X)
- double **cblas_dnrm2** (const int N, const double *X, const int incX)
- void **gsl_blas_dscal** (double alpha, **gsl_vector** *X)
- void **cblas_dscal** (const int N, const double alpha, double *X, const int incX)
- int **gsl_blas_dgemv** (**CBLAS_TRANSPOSE_t** TransA, double alpha, const **gsl_matrix** *A, const **gsl_vector** *X, double beta, **gsl_vector** *Y)
- void **cblas_dgemv** (const enum **CBLAS_ORDER** order, const enum **CBLAS_TRANSPOSE** TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)

7.42 bpmnr/gsl_block.c File Reference

7.42.1 Detailed Description

Definition in file **gsl_block.c**.

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_nr.h>`

Include dependency graph for gsl_block.c:

Functions

- **gsl_block** * **gsl_block_alloc** (const size_t n)
- void **gsl_block_free** (**gsl_block** *b)

7.43 bpmnr/gsl_eigen.c File Reference

7.43.1 Detailed Description

Definition in file **gsl_eigen.c**.

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_nr.h>`

Include dependency graph for gsl_eigen.c:

Functions

- void **chop_small_elements** (**gsl_vector** *d, **gsl_vector** *f)
- void **qrstep** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, **gsl_matrix** *V)
- double **trailing_eigenvalue** (const **gsl_vector** *d, const **gsl_vector** *f)
- void **create_schur** (double d0, double f0, double d1, double *c, double *s)
- void **svd2** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, **gsl_matrix** *V)
- void **chase_out_intermediate_zero** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, size_t k0)
- void **chase_out_trailing_zero** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *V)

7.44 bpmnr/gsl_linalg.c File Reference

7.44.1 Detailed Description

Definition in file **gsl_linalg.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_nr.h>
```

Include dependency graph for gsl_linalg.c:

Functions

- int **gsl_linalg_householder_hm** (double tau, const **gsl_vector** *v, **gsl_matrix** *A)
- int **gsl_linalg_householder_hm1** (double tau, **gsl_matrix** *A)
- void **create_givens** (const double a, const double b, double *c, double *s)
- int **gsl_linalg_bidiag_decomp** (**gsl_matrix** *A, **gsl_vector** *tau_U, **gsl_vector** *tau_V)
- double **gsl_linalg_householder_transform** (**gsl_vector** *v)
- int **gsl_linalg_householder_mh** (double tau, const **gsl_vector** *v, **gsl_matrix** *A)
- int **gsl_linalg_SV_solve** (const **gsl_matrix** *U, const **gsl_matrix** *V, const **gsl_vector** *S, const **gsl_vector** *b, **gsl_vector** *x)
- int **gsl_isnan** (const double x)
- void **chop_small_elements** (**gsl_vector** *d, **gsl_vector** *f)
- void **qrstep** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, **gsl_matrix** *V)
- double **trailing_eigenvalue** (const **gsl_vector** *d, const **gsl_vector** *f)
- void **create_schur** (double d0, double f0, double d1, double *c, double *s)
- void **svd2** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, **gsl_matrix** *V)
- void **chase_out_intermediate_zero** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *U, size_t k0)
- void **chase_out_trailing_zero** (**gsl_vector** *d, **gsl_vector** *f, **gsl_matrix** *V)
- int **gsl_linalg_bidiag_unpack** (const **gsl_matrix** *A, const **gsl_vector** *tau_U, **gsl_matrix** *U, const **gsl_vector** *tau_V, **gsl_matrix** *V, **gsl_vector** *diag, **gsl_vector** *superdiag)

- int **gsl_linalg_bidiag_unpack2** (**gsl_matrix** *A, **gsl_vector** *tau_U, **gsl_vector** *tau_V, **gsl_matrix** *V)
- int **gsl_linalg_SV_decomp** (**gsl_matrix** *A, **gsl_matrix** *V, **gsl_vector** *S, **gsl_vector** *work)

7.45 bpmnr/gsl_matrix.c File Reference

7.45.1 Detailed Description

Definition in file **gsl_matrix.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_nr.h>
```

Include dependency graph for gsl_matrix.c:

Functions

- int **gsl_matrix_swap_columns** (**gsl_matrix** *m, const size_t i, const size_t j)
- **_gsl_vector_view gsl_matrix_column** (**gsl_matrix** *m, const size_t j)
- double **gsl_matrix_get** (const **gsl_matrix** *m, const size_t i, const size_t j)
- void **gsl_matrix_set** (**gsl_matrix** *m, const size_t i, const size_t j, const double x)
- **_gsl_matrix_view gsl_matrix_submatrix** (**gsl_matrix** *m, const size_t i, const size_t j, const size_t n1, const size_t n2)
- **gsl_matrix** * **gsl_matrix_alloc** (const size_t n1, const size_t n2)
- **gsl_matrix** * **gsl_matrix_calloc** (const size_t n1, const size_t n2)
- **_gsl_vector_const_view gsl_matrix_const_row** (const **gsl_matrix** *m, const size_t i)
- **_gsl_vector_view gsl_matrix_row** (**gsl_matrix** *m, const size_t i)
- **_gsl_vector_const_view gsl_matrix_const_column** (const **gsl_matrix** *m, const size_t i)
- void **gsl_matrix_set_identity** (**gsl_matrix** *m)

7.46 bpmnr/gsl_vector.c File Reference

7.46.1 Detailed Description

Definition in file **gsl_vector.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_nr.h>
```

Include dependency graph for gsl_vector.c:

Functions

- _gsl_vector_view **gsl_vector_subvector** (**gsl_vector** *v, size_t offset, size_t n)
- double **gsl_vector_get** (const **gsl_vector** *v, const size_t i)
- void **gsl_vector_set** (**gsl_vector** *v, const size_t i, double x)
- int **gsl_vector_swap_elements** (**gsl_vector** *v, const size_t i, const size_t j)
- **gsl_vector** * **gsl_vector_alloc** (const size_t n)
- **gsl_vector** * **gsl_vector_calloc** (const size_t n)
- _gsl_vector_const_view **gsl_vector_const_subvector** (const **gsl_vector** *v, size_t offset, size_t n)
- void **gsl_vector_free** (**gsl_vector** *v)

7.47 bpmnr/nr_checks.c File Reference

7.47.1 Detailed Description

Definition in file **nr_checks.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_checks.c:

Functions

- int **nr_is_int** (double x)
- int **nr_is_pow2** (unsigned long n)

7.47.2 Function Documentation

7.47.2.1 int nr_is_int (double *x*)

Checks whether the given double is an integer value, handy for doing domain checking to prevent e.g. the function nr_gammln print out "nan" or "inf" values...

For double precision, this check is accurate to 1.0E-323 ... should be enough ;-)

Parameters:

x floating point argument

Returns:

TRUE if argument is indeed an integer value, FALSE if not

Definition at line 21 of file nr_checks.c.

Referenced by nr_gammln().

7.48 bpmnr/nr_complex.c File Reference

7.48.1 Detailed Description

Definition in file **nr_complex.c**.

```
#include "bpm/bpm_nr.h"
```

Include dependency graph for nr_complex.c:

Functions

- **complex_t complex** (double re, double im)
- double **c_real** (**complex_t** z)
- double **c_imag** (**complex_t** z)
- double **c_abs** (**complex_t** z)
- double **c_arg** (**complex_t** z)
- **complex_t c_conj** (**complex_t** z)
- **complex_t c_neg** (**complex_t** z)
- **complex_t c_sum** (**complex_t** z1, **complex_t** z2)
- **complex_t c_diff** (**complex_t** z1, **complex_t** z2)
- **complex_t c_mult** (**complex_t** z1, **complex_t** z2)
- **complex_t c_scale** (double r, **complex_t** z)
- **complex_t c_div** (**complex_t** z1, **complex_t** z2)
- **complex_t c_sqr** (**complex_t** z)
- double **c_norm2** (**complex_t** z)
- **complex_t c_exp** (**complex_t** z)
- **complex_t c_sqrt** (**complex_t** z)
- int **c_isequal** (**complex_t** z1, **complex_t** z2)

7.49 bpmnr/nr_fit.c File Reference

7.49.1 Detailed Description

Definition in file **nr_fit.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_fit.c:

Functions

- int **nr_fit** (double *x, double y[], int ndata, double sig[], int mwt, double *a, double *b, double *siga, double *sigb, double *chi2, double *q)

7.50 bpmnr/nr_four1.c File Reference

7.50.1 Detailed Description

Definition in file **nr_four1.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_four1.c:

Functions

- int **nr_four1** (double data[], unsigned long nn, int isign)

7.51 bpmnr/nr_gammln.c File Reference

7.51.1 Detailed Description

Definition in file **nr_gammln.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_gammln.c:

Functions

- double **nr_gammln** (double xx)

7.52 bpmnr/nr_gammq.c File Reference

7.52.1 Detailed Description

Definition in file **nr_gammq.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_gammq.c:

Functions

- double **nr_gammq** (double a, double x)

7.53 bpmnr/nr_gcf.c File Reference

7.53.1 Detailed Description

Definition in file **nr_gcf.c**.

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_nr.h>`

Include dependency graph for nr_gcf.c:

Functions

- int **nr_gcf** (double *gammcf, double a, double x, double *gln)

7.54 bpmnr/nr_gser.c File Reference

7.54.1 Detailed Description

Definition in file **nr_gser.c**.

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_nr.h>`

Include dependency graph for nr_gser.c:

Functions

- int **nr_gser** (double *gamser, double a, double x, double *gln)

7.55 bpmnr/nr_levmar.c File Reference

7.55.1 Detailed Description

These routines have been written by : and were released under GPL

Manolis Lourakis Institute of Computer Science, Foundation for Research and Technology - Hellas, Heraklion, Crete, Greece

//

Levenberg - Marquardt non-linear minimization algorithm Copyright (C) 2004 Manolis Lourakis (`lourakis@ics.forth.gr`) Institute of Computer Science, Foundation for Research & Technology - Hellas Heraklion, Crete, Greece.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

//

Changes: BM. Modified the names of the routines somewhat to have them correspond to the rest of libbpm

Definition in file **nr_levmar.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_levmar.c:

Defines

- #define **__MIN__**(x, y)
- #define **__MAX__**(x, y)

Functions

- void **nr_trans_mat_mat_mult** (double *a, double *b, int n, int m)
- void **nr_fdif_forw_jac_approx** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *hx, double *hxx, double delta, double *jac, int m, int n, void *adata)
- void **nr_fdif_cent_jac_approx** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *hxm, double *hxp, double delta, double *jac, int m, int n, void *adata)
- void **nr_lmchkjac** (void(*func)(double *p, double *hx, int m, int n, void *adata), void(*jacf)(double *p, double *j, int m, int n, void *adata), double *p, int m, int n, void *adata, double *err)
- int **nr_lmcovar** (double *JtJ, double *C, double sumsq, int m, int n)
- int **nr_lmder** (void(*func)(double *p, double *hx, int m, int n, void *adata), void(*jacf)(double *p, double *j, int m, int n, void *adata), double *p, double *x, int m, int n, int itmax, double opts[4], double info[LM_INFO_SZ], double *work, double *covar, void *adata)
- int **nr_lmdif** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *x, int m, int n, int itmax, double opts[5], double info[LM_INFO_SZ], double *work, double *covar, void *adata)
- int **nr_ax_eq_b_LU** (double *A, double *B, double *x, int m)
- int **nr_lmder_bc** (void(*func)(double *p, double *hx, int m, int n, void *adata), void(*jacf)(double *p, double *j, int m, int n, void *adata), double *p, double *x, int m, int n, double *lb, double *ub, int itmax, double opts[4], double info[LM_INFO_SZ], double *work, double *covar, void *adata)
- void **lmbc_dif_func** (double *p, double *hx, int m, int n, void *data)
- void **lmbc_dif_jacf** (double *p, double *jac, int m, int n, void *data)
- int **nr_lmdif_bc** (void(*func)(double *p, double *hx, int m, int n, void *adata), double *p, double *x, int m, int n, double *lb, double *ub, int itmax, double opts[5], double info[LM_INFO_SZ], double *work, double *covar, void *adata)

7.56 bpmnr/nr_median.c File Reference

7.56.1 Detailed Description

Definition in file **nr_median.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_median.c:

Functions

- double **nr_median** (int n, double *arr)

7.57 bpmnr/nr_ran1.c File Reference

7.57.1 Detailed Description

Definition in file **nr_ran1.c**.

```
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_ran1.c:

Functions

- double **nr_ran1** (long *idum)

7.58 bpmnr/nr_rangauss.c File Reference

7.58.1 Detailed Description

Definition in file **nr_rangauss.c**.

```
#include <stdio.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_rangauss.c:

Functions

- double **nr_rangauss** (double mean, double std_dev)

7.59 bpmnr/nr_ranuniform.c File Reference

7.59.1 Detailed Description

Definition in file **nr_ranuniform.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_ranuniform.c:

Functions

- double **nr_ranuniform** (double lower, double upper)

7.60 bpmnr/nr_realft.c File Reference

7.60.1 Detailed Description

Definition in file **nr_realft.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_realft.c:

Functions

- int **nr_realft** (double data[], unsigned long n, int isign)

7.61 bpmnr/nr_seed.c File Reference

7.61.1 Detailed Description

Definition in file **nr_seed.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_seed.c:

Functions

- int **nr_seed** (long seed)

Variables

- long **bpm_rseed**

7.61.2 Variable Documentation

7.61.2.1 long bpm_rseed

the global random seed variable

Definition at line 9 of file nr_seed.c.

7.62 bpmnr/nr_select.c File Reference

7.62.1 Detailed Description

Definition in file **nr_select.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for nr_select.c:

Functions

- double **nr_select** (int k, int n, double *org_arr)

7.63 bpmorbit/bpm_orbit.h File Reference

7.63.1 Detailed Description

libbpm orbit generation routines

This header contains beam oribit generation routines, so this includes also calibration scans etc...

Definition in file **bpm_orbit.h**.

```
#include <math.h>
#include <bpm/bpm_defs.h>
#include <bpm/bpm_interface.h>
```

Include dependency graph for bpm_orbit.h:

Data Structures

- struct **v3**
- struct **m33**

Functions

- EXTERN int **generate_bpm_orbit** (**beamconf_t** *beam, **bpmconf_t** *bpm)
- EXTERN int **generate_corr_scan** (**bpmconf_t** *bpm, **beamconf_t** *beam, int num_evts, int num_steps, double angle_range, double angle, double z_pos)
- EXTERN int **generate_mover_scan** (**beamconf_t** *beam, int num_evts, int num_steps, double mover_range, double angle)
- void **v_copy** (struct **v3** *v1, struct **v3** *v2)
- double **v_mag** (struct **v3** *v1)
- void **v_scale** (struct **v3** *v1, double dscale)
- void **v_norm** (struct **v3** *v1)
- void **v_matmult** (struct **m33** *m1, struct **v3** *v1)
- void **v_add** (struct **v3** *v1, struct **v3** *v2)
- void **v_sub** (struct **v3** *v1, struct **v3** *v2)
- double **v_dot** (struct **v3** *v1, struct **v3** *v2)
- void **v_cross** (struct **v3** *v1, struct **v3** *v2)
- void **v_print** (struct **v3** *v1)
- void **m_rotmat** (struct **m33** *m1, double alpha, double beta, double gamma)
- void **m_matmult** (struct **m33** *m, struct **m33** *m1, struct **m33** *m2)
- void **m_matadd** (struct **m33** *m1, struct **m33** *m2)
- void **m_print** (struct **m33** *m1)

7.64 bpmorbit/generate_bpm_orbit.c File Reference

7.64.1 Detailed Description

Definition in file **generate_bpm_orbit.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_orbit.h>
```

Include dependency graph for generate_bpm_orbit.c:

Functions

- int **generate_bpm_orbit** (**beamconf_t** *beam, **bpmconf_t** *bpm)

7.65 bpmorbit/generate_corr_scan.c File Reference

7.65.1 Detailed Description

Definition in file **generate_corr_scan.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_orbit.h>
```

Include dependency graph for generate_corr_scan.c:

Functions

- int **generate_corr_scan** (**bpmconf_t** *bpm, **beamconf_t** *beam, int num_evts, int num_steps, double angle_range, double angle, double z_pos)

7.66 bpmorbit/generate_mover_scan.c File Reference

7.66.1 Detailed Description

Definition in file **generate_mover_scan.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_orbit.h>
```

Include dependency graph for generate_mover_scan.c:

Functions

- int **generate_mover_scan** (**beamconf_t** *beam, int num_evts, int num_steps, double mover_range, double angle)

7.67 bpmorbit/vm.c File Reference

7.67.1 Detailed Description

Definition in file **vm.c**.

```
#include <bpm/bpm_orbit.h>
```

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
#include <math.h>
```

Include dependency graph for vm.c:

Functions

- void **v_copy** (struct **v3** *v1, struct **v3** *v2)
- double **v_mag** (struct **v3** *v1)
- void **v_scale** (struct **v3** *v1, double dscale)
- void **v_norm** (struct **v3** *v1)
- void **v_matmult** (struct **m33** *m1, struct **v3** *v1)
- void **v_add** (struct **v3** *v1, struct **v3** *v2)
- void **v_sub** (struct **v3** *v1, struct **v3** *v2)
- double **v_dot** (struct **v3** *v1, struct **v3** *v2)
- void **v_cross** (struct **v3** *v1, struct **v3** *v2)
- void **v_print** (struct **v3** *v1)
- void **m_rotmat** (struct **m33** *m1, double alpha, double beta, double gamma)

- void **m_matmult** (struct **m33** *m, struct **m33** *m1, struct **m33** *m2)
- void **m_matadd** (struct **m33** *m1, struct **m33** *m2)
- void **m_print** (struct **m33** *m1)

7.68 bpmprocess/add_scalar_waveform.c File Reference

7.68.1 Detailed Description

Definition in file **add_scalar_waveform.c**.

```
#include <stdio.h>
#include <stdlib.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for add_scalar_waveform.c:

Functions

- int **add_scalar_waveform** (double *wf, int ns, double add)

7.69 bpmprocess/basic_stats.c File Reference

7.69.1 Detailed Description

Definition in file **basic_stats.c**.

```
#include <math.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for basic_stats.c:

Functions

- int **basic_stats** (int *wf, int ns, int range, int nbits, double *offset, double *rms, int *max, int *min, int *unsat_sample)

7.70 bpmprocess/bpm_process.h File Reference

7.70.1 Detailed Description

libbpm main processing routines

This header contains the definitions for libbpm's main BPM processing routines

Definition in file **bpm_process.h**.

```
#include <float.h>
#include <math.h>
#include <bpm/bpm_defs.h>
#include <bpm/bpm_interface.h>
```

Include dependency graph for bpm_process.h:

Defines

- #define **PROC_DEFAULT**
- #define **PROC_DO_FFT**
- #define **PROC_DO_FIT**
- #define **PROC_DO_DDC**
- #define **PROC_DDC_CALIBFREQ**
- #define **PROC_DDC_CALIBTDECAY**
- #define **PROC_DDC_FITFREQ**
- #define **PROC_DDC_FITTDECAY**
- #define **PROC_DDC_FFTFREQ**
- #define **PROC_DDC_FFTTDECAY**
- #define **PROC_DDC_STOREFULL**
- #define **PROC_FIT_DDC**

Functions

- EXTERN int **process_diode** (**bpmconf_t** *, **bpmsignal_t** *, **bpmproc_t** *)
- EXTERN int **process_waveform** (enum **bpmtype_t** type, **bpmconf_t** *bpm, **bpmcalib_t** *cal, **bpmsignal_t** *sig, **bpmproc_t** *proc, **bpmproc_t** *trig, unsigned int mode)
- EXTERN int **process_monopole** (**bpmconf_t** *bpm, **bpmcalib_t** *cal, **bpmsignal_t** *sig, **bpmproc_t** *proc, **bpmproc_t** *trig, unsigned int mode)
- EXTERN int **process_dipole** (**bpmconf_t** *bpm, **bpmcalib_t** *cal, **bpmsignal_t** *sig, **bpmproc_t** *proc, **bpmproc_t** *trig, **bpmproc_t** *ref, unsigned int mode)
- EXTERN int **fit_waveform** (int *wf, int ns, double t0, double fs, double i_freq, double i_tdecay, double i_amp, double i_phase, double *freq, double *tdecay, double *amp, double *phase)
- EXTERN int **fit_diodepulse** (int *wf, int ns, double fs, double *t0)
- EXTERN int **fit_ddc** (double *ddc, int ns, double *tdecay)
- EXTERN int **fit_fft_prepare** (double **fft, int ns, double fs, int *n1, int *n2, double *amp, double *freq, double *fwhm)
- EXTERN int **fit_fft** (double **fft, int ns, double fs, double *freq, double *tdecay, double *A, double *C)
- EXTERN int **fft_waveform** (int *wf, int ns, double **fft)
- EXTERN int **fft_waveform_double** (double *wf, int ns, double **fft)
- EXTERN int **handle_saturation** (int *wf, int ns, int imax, int nbits, int threshold, int *iunsat)
- EXTERN int **downmix_waveform** (double *wf, int ns, double fs, double freq, double t0, double **out)

- EXTERN int **ddc_gaussfilter_step** (double **ddc, int ns, double fs, int istart, int istop, double tfilter, double filtBW, double *out)
- EXTERN int **ddc_gaussfilter** (double **ddc, int ns, double fs, double filtBW, double epsFilt, double **out)
- EXTERN int **ddc_waveform** (int *wf, int ns, int nbits, double fs, double t0, double freq, double tdecay, double filtBW, double epsFilt, double **out)
- EXTERN int **ddc_sample_waveform** (int *wf, int ns, int nbits, double fs, double t0, double t0Offset, double freq, double tdecay, double filtBW, double epsFilt, double *amp, double *phase)
- EXTERN int **get_pedestal** (int *wf, int ns, int range, double *offset, double *rms)
- EXTERN int **basic_stats** (int *wf, int ns, int range, int nbits, double *offset, double *rms, int *max, int *min, int *unsat_sample)
- EXTERN int **int_to_double_waveform** (double *wf_double, int *wf_int, int ns)
- EXTERN int **copy_waveform** (double *wf_src, double *wf_dst, int ns)
- EXTERN int **add_scalar_waveform** (double *wf, int ns, double add)
- EXTERN int **mult_scalar_waveform** (double *wf, int ns, double mult)
- EXTERN int **mult_waveform** (double *wf1, double *wf2, int ns)
- EXTERN int **get_t0** (int *wf, int ns, double fs, double *t0)
- EXTERN int **get_IQ** (double amp, double phase, double refamp, double refphase, double *Q, double *I)
- EXTERN int **get_pos** (double Q, double I, double IQphase, double posscale, double *pos)
- EXTERN int **get_slope** (double Q, double I, double IQphase, double slopescale, double *slope)
- EXTERN int **time_to_sample** (double fs, int ns, double t, int *iS)
- EXTERN int **sample_to_time** (double fs, int ns, int iS, double *t)
- EXTERN int **freq_to_sample** (double fs, int ns, double f, int *iS)
- EXTERN int **sample_to_freq** (double fs, int ns, int iS, double *f)

7.71 bpmprocess/copy_waveform.c File Reference

7.71.1 Detailed Description

Definition in file **copy_waveform.c**.

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for copy_waveform.c:

Functions

- int **copy_waveform** (double *wf_dst, double *wf_src, int ns)

7.72 bpmprocess/ddc_gaussfilter.c File Reference

7.72.1 Detailed Description

Definition in file **ddc_gaussfilter.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for ddc_gaussfilter.c:

Functions

- int **ddc_gaussfilter** (double **ddc, int ns, double fs, double filtBW, double epsFilt, double **out)

7.73 bpmprocess/ddc_gaussfilter_step.c File Reference

7.73.1 Detailed Description

Definition in file **ddc_gaussfilter_step.c**.

```
#include <math.h>
```

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for ddc_gaussfilter_step.c:

Functions

- int **ddc_gaussfilter_step** (double **ddc, int ns, double fs, int istart, int istop, double tfilter, double filtBW, double *out)

7.74 bpmprocess/ddc_sample_waveform.c File Reference

7.74.1 Detailed Description

Definition in file **ddc_sample_waveform.c**.

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <bpm/bpm_alloc.h>
```

```
#include <bpm/bpm_units.h>
```

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for ddc_sample_waveform.c:

Functions

- int **ddc_sample_waveform** (int *wf, int ns, int nbits, double fs, double t0, double t0Offset, double freq, double tdecay, double filtBW, double epsFilt, double *amp, double *phase)

7.75 bpmprocess/ddc_waveform.c File Reference

7.75.1 Detailed Description

Definition in file **ddc_waveform.c**.

```
#include <stdio.h>
#include <stdlib.h>
#include <bpm/bpm_alloc.h>
#include <bpm/bpm_units.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for ddc_waveform.c:

Functions

- int **ddc_waveform** (int *wf, int ns, int nbits, double fs, double t0, double freq, double tdecay, double filtBW, double epsFilt, double **out)

7.76 bpmprocess/downmix_waveform.c File Reference

7.76.1 Detailed Description

Definition in file **downmix_waveform.c**.

```
#include <math.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for downmix_waveform.c:

Functions

- int **downmix_waveform** (double *wf, int ns, double fs, double freq, double t0, double **out)

7.77 bpmprocess/fft_waveform.c File Reference

7.77.1 Detailed Description

Definition in file **fft_waveform.c**.

```
#include <stdio.h>
#include <stdlib.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for fft_waveform.c:

Functions

- int **fft_waveform_double** (double *wf, int ns, double **fft)
- int **fft_waveform** (int *intwf, int ns, double **fft)

7.78 bpmprocess/fit_ddc.c File Reference

7.78.1 Detailed Description

Definition in file **fit_ddc.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for fit_ddc.c:

Functions

- int **fit_ddc** (double *ddc, int ns, double *tdecay)

7.79 bpmprocess/fit_diodepulse.c File Reference

7.79.1 Detailed Description

Definition in file **fit_diodepulse.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for fit_diodepulse.c:

Functions

- int **fit_diodepulse** (int *wf, int ns, double fs, double *t0)

7.80 bpmprocess/fit_fft.c File Reference

7.80.1 Detailed Description

Definition in file **fit_fft.c**.

```
#include <stdio.h>
#include <bpm/bpm_alloc.h>
#include <bpm/bpm_nr.h>
#include <bpm/bpm_units.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for fit_fft.c:

Defines

- #define **FIT_MAX_ITER**
- #define **FIT_WINDOW_FACTOR**

Functions

- void **fcnlorjac** (double *p, double *ljac, int np, int ns, void *a)
- void **fcnlor** (double *p, double *lor, int np, int ns, void *a)
- int **fit_fft_prepare** (double **fft, int ns, double fs, int *n1, int *n2, double *amp, double *freq, double *fwhm)
- int **fit_fft** (double **fft, int ns, double fs, double *freq, double *tdecay, double *A, double *C)

7.80.2 Function Documentation

7.80.2.1 void fcnlor (double * *p*, double * *lor*, int *np*, int *ns*, void * *a*)

Definition at line 50 of file fit_fft.c.

Referenced by fit_fft().

7.81 bpmprocess/fit_waveform.c File Reference

7.81.1 Detailed Description

Definition in file **fit_waveform.c**.

```
#include <bpm/bpm_nr.h>
```

```
#include <bpm/bpm_alloc.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for fit_waveform.c:

Defines

- #define **FIT_MAX_ITER**
- #define **FIT_AMP**
- #define **FIT_PHASE**
- #define **FIT_FREQ**
- #define **FIT_TDECAY**
- #define **FIT_T0**
- #define **FIT_FS**

Functions

- void **fcnwfjac** (double *par, double *jac, int npars, int ns, void *a)
- void **fcnwf** (double *par, double *sinwf, int npars, int ns, void *a)
- int **fit_waveform** (int *wf, int ns, double t0, double fs, double i_freq, double i_tdecay, double i_amp, double i_phase, double *freq, double *tdecay, double *amp, double *phase)

7.81.2 Function Documentation

7.81.2.1 void fcnwf (double * *par*, double * *sinwf*, int *npars*, int *ns*, void * *a*)

The fitfunction, being simply the waveform, setup for the additional data array xval[0] = t0 xval[1] = the sampling frequency

Definition at line 62 of file fit_waveform.c.

References FIT_AMP, FIT_FREQ, FIT_FS, FIT_PHASE, FIT_T0, FIT_TDECAY, and sample_to_time().

Referenced by fit_waveform().

7.82 bpmprocess/freq_to_sample.c File Reference

7.82.1 Detailed Description

Definition in file **freq_to_sample.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for freq_to_sample.c:

Functions

- int **freq_to_sample** (double fs, int ns, double f, int *iS)

7.83 bpmprocess/get_IQ.c File Reference

7.83.1 Detailed Description

Definition in file **get_IQ.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for get_IQ.c:

Functions

- int **get_IQ** (double amp, double phase, double refamp, double refphase, double *Q, double *I)

7.84 bpmprocess/get_pedestal.c File Reference

7.84.1 Detailed Description

Definition in file **get_pedestal.c**.

```
#include <math.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for get_pedestal.c:

Functions

- int **get_pedestal** (int *wf, int ns, int range, double *offset, double *rms)

7.85 bpmprocess/get_pos.c File Reference

7.85.1 Detailed Description

Definition in file **get_pos.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for get_pos.c:

Functions

- int **get_pos** (double Q, double I, double IQphase, double posscale, double *pos)

7.86 bpmprocess/get_slope.c File Reference

7.86.1 Detailed Description

Definition in file **get_slope.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for get_slope.c:

Functions

- int **get_slope** (double Q, double I, double IQphase, double slopescale, double *slope)

7.87 bpmprocess/get_t0.c File Reference

7.87.1 Detailed Description

Declared two helper routines which find the start and end samples for the fit...

Definition in file **get_t0.c**.

```
#include <stdlib.h>
#include <math.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for get_t0.c:

Functions

- void **find_t0_startfit** (int *wf, double ped, int peak_sample, double peak_value, double peak_fraction, int *start_sample)
- void **find_t0_endfit** (int *wf, double ped, int peak_sample, double peak_value, double peak_fraction, int *end_sample)
- int **get_t0** (int *wf, int ns, double fs, double *t0)

7.88 bpmprocess/handle_saturation.c File Reference

7.88.1 Detailed Description

Definition in file **handle_saturation.c**.

```
#include <math.h>
#include <limits.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for handle_saturation.c:

Functions

- int **handle_saturation** (int *wf, int ns, int imax, int nbits, int threshold, int *iunsat)

7.89 bpmprocess/int_to_double_waveform.c File Reference

7.89.1 Detailed Description

Definition in file **int_to_double_waveform.c**.

```
#include <stdio.h>
#include <stdlib.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for int_to_double_waveform.c:

Functions

- int **int_to_double_waveform** (double *wf_double, int *wf_int, int ns)

7.90 bpmprocess/mult_scalar_waveform.c File Reference

7.90.1 Detailed Description

Definition in file **mult_scalar_waveform.c**.

```
#include <stdio.h>
#include <stdlib.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for mult_scalar_waveform.c:

Functions

- int **mult_scalar_waveform** (double *wf, int ns, double mult)

7.91 bpmprocess/mult_waveform.c File Reference

7.91.1 Detailed Description

Definition in file **mult_waveform.c**.

```
#include <stdio.h>
#include <stdlib.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for mult_waveform.c:

Functions

- int **mult_waveform** (double *wf1, double *wf2, int ns)

7.92 bpmprocess/process_diode.c File Reference

7.92.1 Detailed Description

Definition in file **process_diode.c**.

```
#include <stdio.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_process.h>
```

Include dependency graph for process_diode.c:

Functions

- int **process_diode** (**bpmconf_t** *bpm, **bpmsignal_t** *sig, **bpmproc_t** *proc)

7.93 bpmprocess/process_dipole.c File Reference

7.93.1 Detailed Description

Definition in file **process_dipole.c**.

```
#include <stdio.h>
```

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for process_dipole.c:

Functions

- int **process_dipole** (**bpmconf_t** *bpm, **bpmcalib_t** *cal, **bpmsignal_t** *sig, **bpmproc_t** *proc, **bpmproc_t** *trig, **bpmproc_t** *ref, unsigned int mode)

7.94 bpmprocess/process_monopole.c File Reference

7.94.1 Detailed Description

Definition in file **process_monopole.c**.

```
#include <stdio.h>
```

```
#include <bpm/bpm_units.h>
```

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for process_monopole.c:

Functions

- int **process_monopole** (**bpmconf_t** *bpm, **bpmcalib_t** *cal, **bpmsignal_t** *sig, **bpmproc_t** *proc, **bpmproc_t** *trig, unsigned int mode)

7.95 bpmprocess/process_waveform.c File Reference

7.95.1 Detailed Description

Definition in file **process_waveform.c**.

```
#include <stdio.h>
```

```
#include <bpm/bpm_units.h>
```

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for process_waveform.c:

Functions

- int **process_waveform** (enum **bpmtype_t** type, **bpmconf_t** *bpm, **bpmcalib_t** *cal, **bpmsignal_t** *sig, **bpmproc_t** *proc, **bpmproc_t** *trig, unsigned int mode)

7.96 bpmprocess/sample_to_freq.c File Reference

7.96.1 Detailed Description

Definition in file **sample_to_freq.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for sample_to_freq.c:

Functions

- int **sample_to_freq** (double fs, int ns, int iS, double *f)

7.97 bpmprocess/sample_to_time.c File Reference

7.97.1 Detailed Description

Definition in file **sample_to_time.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for sample_to_time.c:

Functions

- int **sample_to_time** (double fs, int ns, int iS, double *t)

7.98 bpmprocess/time_to_sample.c File Reference

7.98.1 Detailed Description

Definition in file **time_to_sample.c**.

```
#include <bpm/bpm_messages.h>
```

```
#include <bpm/bpm_process.h>
```

Include dependency graph for time_to_sample.c:

Functions

- int **time_to_sample** (double fs, int ns, double t, int *iS)

7.99 bpmrf/bpm_rf.h File Reference

7.99.1 Detailed Description

libbpm rf simulation routines

The header file for RF routines

Need to check in how far these routines are redundant, bpmdsp can replace most of the filtering routines here !

Definition in file **bpm_rf.h**.

```
#include <math.h>
```

```
#include <bpm/bpm_defs.h>
```

```
#include <bpm/bpm_interface.h>
```

Include dependency graph for bpm_rf.h:

Functions

- EXTERN int **rf_setup** (int nsamples, double sfreq)
- EXTERN int **rf_rectify** (double **IF)
- EXTERN int **rf_filter** (double **RF, enum **rffiltertype_t** filttype, int nfiltpar, double *pars)
- EXTERN int **rf_butterworthlowpass** (double **RF, int order, double fc)
- EXTERN int **rf_butterworthbandpass** (double **RF, int order, double f0, double BW)
- EXTERN int **rf_butterworthhighpass** (double **RF, int order, double fc)
- EXTERN int **rf_complexFFT** (double **in, double **out, int dir)
- EXTERN int **rf_addLO** (double amp, double lofreq, enum **bpmphase_t** type, double phi0, double d_phi, double **LO)
- EXTERN int **rf_mixer** (double *RF_Re, double *RF_Im, double **LO, double *IF)
- EXTERN int **rf_amplify** (double *RF, double dB)

Variables

- EXTERN int **rf_nsamples**
- EXTERN double **rf_samplefreq**

7.100 bpmrf/rf_addLO.c File Reference

7.100.1 Detailed Description

Definition in file **rf_addLO.c**.

```
#include <bpm/bpm_interface.h>
```

```
#include <bpm/bpm_rf.h>
#include <bpm/bpm_nr.h>
#include <math.h>
```

Include dependency graph for rf_addLO.c:

Functions

- int **rf_addLO** (double amp, double lofreq, enum **bpmphase_t** type, double phi0, double d_phi, double **LO)

7.101 bpmrf/rf_amplify.c File Reference

7.101.1 Detailed Description

Definition in file **rf_amplify.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for rf_amplify.c:

Functions

- int **rf_amplify** (double *RF, double dB)

7.102 bpmrf/rf_butterworthbandpass.c File Reference

7.102.1 Detailed Description

Definition in file **rf_butterworthbandpass.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_alloc.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for rf_butterworthbandpass.c:

Functions

- int **rf_butterworthbandpass** (double **RF, int order, double f0, double BW)

7.103 bpmrf/rf_butterworthhighpass.c File Reference

7.103.1 Detailed Description

Definition in file **rf_butterworthhighpass.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_alloc.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for rf_butterworthhighpass.c:

Functions

- int **rf_butterworthhighpass** (double **RF, int order, double fc)

7.104 bpmrf/rf_butterworthlowpass.c File Reference

7.104.1 Detailed Description

Definition in file **rf_butterworthlowpass.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_alloc.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for rf_butterworthlowpass.c:

Functions

- int **rf_butterworthlowpass** (double **RF, int order, double fc)

7.105 bpmrf/rf_complexFFT.c File Reference

7.105.1 Detailed Description

Definition in file **rf_complexFFT.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_alloc.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for rf_complexFFT.c:

Functions

- int **rf_complexFFT** (double **in, double **out, int dir)

7.106 bpmrf/rf_filter.c File Reference

7.106.1 Detailed Description

Definition in file **rf_filter.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for rf_filter.c:

Functions

- int **rf_filter** (double **RF, enum **rffiltertype_t** filttype, int nfiltpar, double *pars)

7.107 bpmrf/rf_mixer.c File Reference

7.107.1 Detailed Description

Definition in file **rf_mixer.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for rf_mixer.c:

Functions

- int **rf_mixer** (double *RF_Re, double *RF_Im, double **LO, double *IF)

7.108 bpmrf/rf_rectify.c File Reference

7.108.1 Detailed Description

Definition in file **rf_rectify.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_units.h>
```

Include dependency graph for rf_rectify.c:

Functions

- int **rf_rectify** (double **IF)

7.109 bpmrf/rf_setup.c File Reference

7.109.1 Detailed Description

Definition in file **rf_setup.c**.

```
#include <bpm/bpm_interface.h>
#include <bpm/bpm_units.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for rf_setup.c:

Functions

- int **rf_setup** (int nsamples, double sfreq)

Variables

- int **rf_nsamples**
- double **rf_samplefreq**

7.110 bpmsimulation/add_amplnoise.c File Reference

7.110.1 Detailed Description

Definition in file **add_amplnoise.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_nr.h>
```

Include dependency graph for add_amplnoise.c:

Functions

- int **add_amplnoise** (double amplnoise, double *IF_Re, double *IF_Im)

7.111 bpmsimulation/add_excitation.c File Reference

7.111.1 Detailed Description

Definition in file **add_excitation.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
#include <math.h>
```

Include dependency graph for add_excitation.c:

Functions

- int **add_excitation** (double ttrig, double *RF)

7.112 bpmsimulation/add_wave.c File Reference

7.112.1 Detailed Description

Definition in file **add_wave.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for add_wave.c:

Functions

- int **add_wave** (double amp, double phase, double freq, double ttrig, double tdecay, double **RF)

7.113 bpmsimulation/add_waveforms.c File Reference

7.113.1 Detailed Description

Definition in file **add_waveforms.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for add_waveforms.c:

Functions

- int **add_waveforms** (double *RF, double *RFadd, double factor)

7.114 bpmsimulation/bpm_simulation.h File Reference

7.114.1 Detailed Description

libbpm waveform simulation routines

This header contains the definitions for the libbpm RF waveform simulation routines

Definition in file **bpm_simulation.h**.

```
#include <math.h>
#include <bpm/bpm_defs.h>
#include <bpm/bpm_interface.h>
```

Include dependency graph for bpm_simulation.h:

Functions

- EXTERN int **generate_monopole** (**bpmconf_t** *, **beamconf_t** *, **bpmsignal_t** *)
- EXTERN int **generate_dipole** (**bpmconf_t** *, **beamconf_t** *, **bpmsignal_t** *)
- EXTERN int **generate_diode** (**bpmconf_t** *, **beamconf_t** *, **bpmsignal_t** *)
- EXTERN int **get_monopole_response** (double bunchcharge, double chargesens, double arrivaltime, double cavityfreq, double *amp, double *phase)
- EXTERN int **get_dipole_response** (double bunchcharge, double chargesens, double pos, double possens, double tilt, double tiltsens, double arrivaltime, double cavityfreq, double *amp, double *phase)
- EXTERN int **get_dipole_amp** (double bunchcharge, double bunchlength, double pos, double possens, double slope, double slopesens, double tilt, double tiltsens, double *amp, double *phase)
- EXTERN int **get_monopole_amp** (double bunchcharge, double bunchlength, double chargesens, double *amp, double *phase)
- EXTERN int **add_excitation** (double ttrig, double *RF)
- EXTERN int **simple_wave** (double amp, double phase, double ttrig, double freq, double tdecay, double ped, double ampnoise, double phasenoise, double fs, int nbits, int *wf, int ns)
- EXTERN int **simple_tone** (double amp, double phase, double freq, double ped, double ampnoise, double phasenoise, double fs, int nbits, int *wf, int ns)
- EXTERN int **add_wave** (double amp, double phase, double freq, double ttrig, double tdecay, double **RF)
- EXTERN int **add_waveforms** (double *RF, double *RFadd, double factor)
- EXTERN int **reset_complex_wave** (double **RF)
- EXTERN int **reset_simple_wave** (int ns, double *wf)
- EXTERN int **get_real_part** (double **RF, double *wf)
- EXTERN int **get_imaginary_part** (double **RF, double *wf)
- EXTERN int **get_amplitude** (double **RF, double *wf)
- EXTERN int **get_phase** (double **RF, double *wf)

- EXTERN int **get_complex_from_ReIm** (double *RF_Re, double *RF_Im, double **RF)
- EXTERN int **get_complex_from_AmpPhi** (double *Amp, double *Phi, double **RF)
- EXTERN int **add_amplnoise** (double amplnoise, double *IF_Re, double *IF_Im)
- EXTERN int **digitise** (double *IF, int nbits, double fs, double range_min, double range_max, int ns, int *wf)

7.115 bpmsimulation/digitise.c File Reference

7.115.1 Detailed Description

Definition in file **digitise.c**.

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_simulation.h>`

`#include <bpm/bpm_rf.h>`

`#include <bpm/bpm_nr.h>`

Include dependency graph for digitise.c:

Functions

- int **digitise** (double *IF, int nbits, double fs, double range_min, double range_max, int ns, int *wf)

7.116 bpmsimulation/generate_diode.c File Reference

7.116.1 Detailed Description

Definition in file **generate_diode.c**.

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_simulation.h>`

`#include <bpm/bpm_rf.h>`

`#include <bpm/bpm_alloc.h>`

Include dependency graph for generate_diode.c:

Functions

- int **generate_diode** (**bpmconf_t** *bpm, **beamconf_t** *beam, **bpmsignal_t** *sig)

7.117 bpmsimulation/generate_dipole.c File Reference

7.117.1 Detailed Description

Definition in file **generate_dipole.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_alloc.h>
```

Include dependency graph for generate_dipole.c:

Functions

- int **generate_dipole** (**bpmconf_t** *bpm, **beamconf_t** *beam, **bpmsignal_t** *sig)

7.118 bpmsimulation/generate_monopole.c File Reference

7.118.1 Detailed Description

Definition in file **generate_monopole.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_alloc.h>
```

Include dependency graph for generate_monopole.c:

Functions

- int **generate_monopole** (**bpmconf_t** *bpm, **beamconf_t** *beam, **bpmsignal_t** *sig)

7.119 bpmsimulation/generate_noise.c File Reference

7.119.1 Detailed Description

Definition in file **generate_noise.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
```

Include dependency graph for generate_noise.c:

Functions

- int **generate_noise** (**bpmconf_t** *bpm, **beamconf_t** *beam, **bpmsignal_t** *sig)

7.119.2 Function Documentation

7.119.2.1 int generate_noise (bpmconf_t * *bpm*, beamconf_t * *beam*, bpmsignal_t * *sig*)

NOT IMPLEMENTED YET !

Definition at line 12 of file generate_noise.c.

References bpm_error().

7.120 bpmsimulation/get_amplitude.c File Reference

7.120.1 Detailed Description

Definition in file **get_amplitude.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for get_amplitude.c:

Functions

- int **get_amplitude** (double **RF, double *wf)

7.121 bpmsimulation/get_complex_from_AmpPhi.c File Reference

7.121.1 Detailed Description

Definition in file **get_complex_from_AmpPhi.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for get_complex_from_AmpPhi.c:

Functions

- int **get_complex_from_AmpPhi** (double *Amp, double *Phi, double **RF)

7.122 bpmsimulation/get_complex_from_ReIm.c File Reference

7.122.1 Detailed Description

Definition in file **get_complex_from_ReIm.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for get_complex_from_ReIm.c:

Functions

- int **get_complex_from_ReIm** (double *RF_Re, double *RF_Im, double **RF)

7.123 bpmsimulation/get_dipole_amp.c File Reference

7.123.1 Detailed Description

Definition in file **get_dipole_amp.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <math.h>
```

Include dependency graph for get_dipole_amp.c:

Functions

- int **get_dipole_amp** (double bunchcharge, double bunchlength, double pos, double possens, double slope, double slopesens, double tilt, double tiltsens, double *amp, double *phase)

7.124 bpmsimulation/get_dipole_response.c File Reference

7.124.1 Detailed Description

Definition in file **get_dipole_response.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
```

Include dependency graph for get_dipole_response.c:

Functions

- int **get_dipole_response** (double bunchcharge, double chargesens, double pos, double possens, double tilt, double tiltsens, double arrivaltime, double cavityfreq, double *amp, double *phase)

7.125 bpmsimulation/get_imaginary_part.c File Reference

7.125.1 Detailed Description

Definition in file **get_imaginary_part.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for get_imaginary_part.c:

Functions

- int **get_imaginary_part** (double **RF, double *wf)

7.126 bpmsimulation/get_monopole_amp.c File Reference

7.126.1 Detailed Description

Definition in file **get_monopole_amp.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <math.h>
```

Include dependency graph for get_monopole_amp.c:

Functions

- int **get_monopole_amp** (double bunchcharge, double bunchlength, double chargesens, double *amp, double *phase)

7.127 bpmsimulation/get_monopole_response.c File Reference

7.127.1 Detailed Description

Definition in file **get_monopole_response.c**.

```
#include <bpm/bpm_messages.h>
```

`#include <bpm/bpm_simulation.h>`

Include dependency graph for get_monopole_response.c:

Functions

- int **get_monopole_response** (double bunchcharge, double chargesens, double arrivaltime, double cavityfreq, double *amp, double *phase)

7.128 bpmsimulation/get_phase.c File Reference

7.128.1 Detailed Description

Definition in file **get_phase.c**.

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_simulation.h>`

`#include <bpm/bpm_rf.h>`

Include dependency graph for get_phase.c:

Functions

- int **get_phase** (double **RF, double *wf)

7.129 bpmsimulation/get_real_part.c File Reference

7.129.1 Detailed Description

Definition in file **get_real_part.c**.

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_simulation.h>`

`#include <bpm/bpm_rf.h>`

Include dependency graph for get_real_part.c:

Functions

- int **get_real_part** (double **RF, double *wf)

7.130 bpmsimulation/reset_complex_wave.c File Reference

7.130.1 Detailed Description

Definition in file **reset_complex_wave.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for reset_complex_wave.c:

Functions

- int **reset_complex_wave** (double **RF)

7.131 bpmsimulation/reset_simple_wave.c File Reference

7.131.1 Detailed Description

Definition in file **reset_simple_wave.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
```

Include dependency graph for reset_simple_wave.c:

Functions

- int **reset_simple_wave** (int ns, double *wf)

7.132 bpmsimulation/simple_tone.c File Reference

7.132.1 Detailed Description

Definition in file **simple_tone.c**.

```
#include <math.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
#include <bpm/bpm_units.h>
#include <bpm/bpm_simulation.h>
```

Include dependency graph for simple_tone.c:

Functions

- int **simple_tone** (double amp, double phase, double freq, double ped, double ampnoise, double phasenoise, double fs, int nbits, int *wf, int ns)

7.133 bpmsimulation/simple_wave.c File Reference

7.133.1 Detailed Description

Definition in file **simple_wave.c**.

```
#include <math.h>
#include <bpm/bpm_messages.h>
#include <bpm/bpm_nr.h>
#include <bpm/bpm_units.h>
#include <bpm/bpm_simulation.h>
```

Include dependency graph for simple_wave.c:

Functions

- int **simple_wave** (double amp, double phase, double ttrig, double freq, double tdecay, double ped, double ampnoise, double phasenoise, double fs, int nbits, int *wf, int ns)

Index