libbpm

0.3

Generated by Doxygen 1.5.6

Wed Jun 25 17:31:48 2008

Contents

1 libbpm

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

libbpm is a C-library which contains low level beam position monitor (BPM) signal processing routines. It's aim is to form a complete set of routines needed to handle RF Cavity BPM data, from digital down-mixing, sampling, calibrating analysing and simulating BPM data. This library has been developed in the context of the BPM work done by the accelerator physics groups at University College London, Royal Holloway University of London and the University of Cambridge (UK) (2006-2007)

The library consists out of a set of submodules which take care of different parts of the BPM signal handling. There are modules for BPM processing, calibration, simulation, general waveform handling, some numerical routines, memory management etc...

The library is licenced under the **GNU General Public License v2.** (p. 3)

1.2 Documentation structure

The documentation for this library is generated using doyxygen. For each module the documentation is contained in it's respective header file :

- **The waveform handling module** (p. 84)
- **The digital signal processing module** (p. 48)
- **The BPM processing module** (p. 67)

1.3 Compilation

The compilation of the libbpm structure is defined using the GNU autotools. Therefore making it portable under most unix flavours and MacOS as well as windows (see futher).

1.3.1 Compilation under Linux/Unix/MacOS

For compilation under any unix flavour, please execute the standart sequence of ./configure , `make`, and `make` install. The default options for the configure script apply.

If you have extracted the library from CVS, then you will have to generate the build scripts. the `autogen.sh` script takes care of that. Run it and afterwards you can simply execute the same steps as above.

1.3.2 Note on Compilation under Windows

This is a remnant from libespec, need to retest this and write proper documentation on it, but for what it's worth... here goes :

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.

1.4 Using libbpm in your programs

libbpm is a standalone plain C library. Care has been taken to not have to use special compiler options e.g. the library avoids having to be C99 compliant by implementing it's own complex data type, rounding function etc.. So it should be fairly portable to most platforms.

To use libbpm in your makefiles for your project, a convenient script has been created which automatically gives you the correct compiler options and library locations. See this makefile example on how to use the script `libbpm-config`

```
#Example makefile that uses libbpm and ROOT (hey.. why not :D !)

SRC         = main.cpp subroutine.cpp

ROOT_LIBS   = $(shell root-config --libs)
ROOT_CFLAGS = $(shell root-config --cflags)

BPM_LIBS    = $(shell libbpm-config --libs)
BPM_CFLAGS  = $(shell libbpm-config --cflags)

CPP         = g++
CPPFLAGS    = -O3 -Wall -fPIC -fno-strict-aliasing $(BPM_CFLAGS) $(ROOT_CFLAGS)
```

```
LD          = g++
LDFLAGS     = $(BPM_LIBS) $(ROOT_LIBS)

OBJ         = $(SRC:.cpp=.o)

#suffix rules
.SUFFIXES: .cpp .o
.cpp.o:
        $(CPP) $(CPPFLAGS) -c $<

#build rules
.PHONY: all
all: program

program: $(OBJ)
        $(LD) $(LDFLAGS) $^ -o $@
```

You can use the `-help` option of `libbpm-config` to display it's options :

```
[linux] ~/libbpm $ libbpm-config --help
Usage: libbpm-config [OPTION]

Known values for OPTION are:

  --prefix              show libbpm installation prefix
  --libs                print library linking information
  --cflags              print pre-processor and compiler flags
  --help                display this help and exit
  --version             output version information
```

2 GNU General Public License, v2

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software– to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other

pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER

3 Module Index

3.1 Modules

Here is a list of all modules:

4 Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

5 File Index

5.1 File List

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

6 Module Documentation

6.1 Analysis routines

6.1.1 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 **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)

6.1.2 Define Documentation

6.1.2.1 #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().

6.1.2.2 #define BPM_BAD_EVENT

A bad event

Definition at line 29 of file bpm_analysis.h.

6.1.2.3 #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().

6.1.2.4 #define ANA_SVD_NOTILT

Don't include tilts in the SVD

Definition at line 32 of file bpm_analysis.h.

6.1.3 Function Documentation

6.1.3.1 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.

6.1.3.2 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** (p. 123) 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_matrix_set(), gsl_vector_get(), and gsl_vector_set().

6.1.3.3 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. 13)

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, bpmproc::ddc_pos, and bpmproc::ddc_-slope.

6.1.3.4 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.

6.1.4 Variable Documentation

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

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

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

6.2 Calibration routines

6.2.1 Detailed Description

Files

- file **bpm_calibration.h**
  calibration routines
- file **calibrate.c**
- file **setup_calibration.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, **bunchconf_t** *bunch)
- EXTERN int **calibrate** (**bpmconf_t** *bpm, **bunchconf_t** *bunch, **bpmproc_t** *proc, int npulses, **bpmcalib_t** *cal)

6.2.2 Function Documentation

6.2.2.1 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*, bunchconf_t * *bunch*)

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

bunch the returned **bunchconf** (p. 138) 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 bunchconf::bpmposition.

6.2.2.2 EXTERN int calibrate (bpmconf_t ∗ *bpm*, bunchconf_t ∗ *bunch*, 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

bunch An array of bunch 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().

6.3 Beam orbit generation

6.3.1 Detailed Description

Files

- file **bpm_orbit.h**

  libbpm orbit generation routines

- file **get_bpmhit.c**
- file **vm.c**

Data Structures

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

Functions

- EXTERN double **get_rbend** (double e, double B, double l, double p)
- EXTERN double **get_sbend** (double e, double B, double l, double p)
- EXTERN int **get_bpmhit** (**bunchconf_t** ∗bunch, **bpmconf_t** ∗bpm)
- EXTERN int **get_bpmhits** (**beamconf_t** ∗beam, **bpmconf_t** ∗bpm)
- 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)

6.3.2 Function Documentation

6.3.2.1 EXTERN double get_rbend (double *e*, double *B*, double *l*, double *p*)

Get the bending angle through a rectangular bending magnet

Parameters:

e the particle's charge in units of e, take sign into account !

B the magnetic field in Tesla
l the length of the magnet in meter
p the momentum of the particle in GeV

Returns:

the bending angle

get_rbend.c

Definition at line 12 of file get_bend.c.

6.3.2.2 EXTERN double get_sbend (double *e*, double *B*, double *l*, double *p*)

Get the bending angle through a sector bending magnet

Parameters:

e the particle's charge in units of e, take sign into account !
B the magnetic field in Tesla
l the sector length of the magnet in meter
p the momentum of the particle in GeV

Returns:

the bending angle

Definition at line 17 of file get_bend.c.

6.3.2.3 EXTERN int get_bpmhit (bunchconf_t * *bunch*, bpmconf_t * *bpm*)

Get the bunch hit in the local BPM coordinate frame

Parameters:

bunch the bunch structure
bpm the bpm config

Definition at line 34 of file get_bpmhit.c.

References bpm_error(), bunchconf::bpmposition, bunchconf::bpmslope, bunchconf::bpmtilt, bpmconf::geom_pos, bpmconf::geom_tilt, m_rotmat(), bunchconf::position, bunchconf::slope, v_add(), v_copy(), v_cross(), v_dot(), v_matmult(), v_scale(), v_sub(), v3::x, v3::y, and v3::z.

Referenced by get_bpmhits().

6.3.2.4 EXTERN int get_bpmhits (beamconf_t * *beam*, bpmconf_t * *bpm*)

Calls get_bpmhit for every bunch in the beam...

Parameters:

beam the beam structure
bpm the bpm config

Definition at line 9 of file get_bpmhit.c.

References bpm_error(), beamconf::bunch, get_bpmhit(), and beamconf::nbunches.

6.3.2.5 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 get_bpmhit().

6.3.2.6 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().

6.3.2.7 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 get_bpmhit(), and v_norm().

6.3.2.8 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().

6.3.2.9 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 get_bpmhit().

6.3.2.10 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 get_bpmhit().

6.3.2.11 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 get_bpmhit().

6.3.2.12 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 get_bpmhit(), and v_mag().

6.3.2.13 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 get_bpmhit().

6.3.2.14 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.

6.3.2.15 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 get_bpmhit().

6.3.2.16 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().

6.3.2.17 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.

6.3.2.18 void m_print (struct m33 * *m1*)

Print 3x3 matrix m1 to stdout

Definition at line 151 of file vm.c.

References m33::e.

6.4 Front-end interface

6.4.1 Detailed Description

Files

- file **bpm_interface.h**
  Front end interface structure definitions and handlers.

Data Structures

- struct **bpmconf**
- struct **bpmcalib**
- struct **bpmproc**
- struct **beamconf**
- struct **bunchconf**
- struct **bpmmode**
- struct **rfmodel**

Typedefs

- typedef struct **bpmconf bpmconf_t**
- typedef struct **bpmcalib bpmcalib_t**
- typedef struct **bpmproc bpmproc_t**
- typedef struct **beamconf beamconf_t**
- typedef struct **bunchconf bunchconf_t**
- typedef struct **bpmmode bpmmode_t**
- typedef struct **rfmodel rfmodel_t**
- typedef enum **triggertype triggertype_t**

Enumerations

- enum **bpmtype_t** { **diode**, **monopole**, **dipole** }
- enum **triggertype** { **positive**, **negative**, **bipolar** }
- enum **bpmpol_t** { **horiz**, **vert** }
- enum **bpmphase_t** { **randomised**, **locked** }

Variables

- EXTERN int **bpm_verbose**
- EXTERN int **libbpm_evtnum**

6.4.2 Typedef Documentation

6.4.2.1 typedef struct bpmconf bpmconf_t

type definition for BPM configuration

Definition at line 73 of file bpm_interface.h.

6.4.2.2 typedef struct bpmcalib bpmcalib_t

type definition for calibrations

Definition at line 74 of file bpm_interface.h.

6.4.2.3 typedef struct bpmproc bpmproc_t

type definition for processed BPM signals

Definition at line 75 of file bpm_interface.h.

6.4.2.4 typedef struct beamconf beamconf_t

type definition for beam configurations

Definition at line 76 of file bpm_interface.h.

6.4.2.5 typedef struct bunchconf bunchconf_t

type definition for bunch configurations

Definition at line 77 of file bpm_interface.h.

6.4.3 Enumeration Type Documentation

6.4.3.1 enum bpmtype_t

BPM cavity (of better signal) type

Enumerator:

diode rectified bpm signal (trigger pulse)
monopole reference cavity signal (monopole)
dipole position sentivive cavity signal (dipole)

Definition at line 41 of file bpm_interface.h.

6.4.3.2 enum triggertype

Diode behavior type

Enumerator:

positive Positive half-period of the waveform is detected
negative Negative half-period of the waveform is detected
bipolar The both half-periods are detected

Definition at line 50 of file bpm_interface.h.

6.4.3.3 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 59 of file bpm_interface.h.

6.4.3.4 enum bpmphase_t

BPM electronics phase lock type

Enumerator:

randomised unlocked phase
locked locked phase

Definition at line 67 of file bpm_interface.h.

6.4.4 Variable Documentation

6.4.4.1 EXTERN int bpm_verbose

be a bit verbose in libbpm

Definition at line 308 of file bpm_interface.h.

Referenced by get_t0().

6.4.4.2 EXTERN int libbpm_evtnum

the global event number in the processing

Definition at line 309 of file bpm_interface.h.

Referenced by bpm_error(), and bpm_warning().

6.5 Error/warning messages

6.5.1 Detailed Description

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)

6.5.2 Function Documentation

6.5.2.1 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 9 of file bpm_error.c.

References libbpm_evtnum.

Referenced by _expand_complex_polynomial(), add_mode_response(), ana_set_cutfn(), apply_filter(), calibrate(), check_saturation(), complexfft(), complexwf(), complexwf_add(), complexwf_add_ampnoise(), complexwf_add_cwtone(), complexwf_add_dcywave(), complexwf_add_noise(), complexwf_add_phasenoise(), complexwf_bias(), complexwf_compat(), complexwf_copy(), complexwf_copy_new(), complexwf_divide(), complexwf_getamp(), complexwf_getamp_new(), complexwf_getimag(), complexwf_getimag_new(), complexwf_getphase(), complexwf_getphase_new(), complexwf_getreal(), complexwf_getreal_new(), complexwf_multiply(), complexwf_print(), complexwf_reset(), complexwf_scale(), complexwf_setfunction(), complexwf_setimag(), complexwf_setreal(), complexwf_setvalues(), complexwf_subset(), complexwf_subtract(), correct_gain(), create_filter(), create_resonator_representation(), create_splane_representation(), ddc_initialise(), ddc_sample_waveform(), ddc_waveform(), digitise(), doublewf(), doublewf_add(), doublewf_add_ampnoise(), doublewf_add_cwtone(), doublewf_add_dcywave(), doublewf_basic_stats(), doublewf_bias(), doublewf_cast(), doublewf_cast_new(), doublewf_compat(), doublewf_copy(), doublewf_copy_new(), doublewf_derive(), doublewf_divide(), doublewf_getvalue(), doublewf_integrate(), doublewf_multiply(), doublewf_print(), doublewf_resample(), doublewf_reset(), doublewf_scale(), doublewf_setvalues(), doublewf_subset(), doublewf_subtract(), downmix_waveform(), fft_gen_tables(), fft_initialise(), fft_waveform(), filter_impulse_response(), filter_step_response(), fit_fft(), fit_fft_prepare(), fit_waveform(), gaussian_filter_coeffs(), generate_bpmsignal(), generate_diodesignal(), get_bpmhit(), get_bpmhits(), get_IQ(), get_mode_response(), get_pedestal(), get_pos(), get_slope(), get_t0(), gsl_block_alloc(), gsl_matrix_column(), gsl_matrix_submatrix(), gsl_matrix_swap_columns(), gsl_vector_subvector(), intwf(), intwf_add(), intwf_add_ampnoise(), intwf_add_cwtone(), intwf_add_dcywave(), intwf_basic_stats(), intwf_bias(), intwf_cast(), intwf_cast_new(), intwf_compat(), intwf_copy(), intwf_copy_new(), intwf_derive(), intwf_divide(), intwf_getvalue(), intwf_integrate(), intwf_multiply(), intwf_print(), intwf_resample(), intwf_reset(), intwf_scale(), intwf_setvalues(), intwf_subset(), intwf_subtract(), normalise_filter(), nr_fit(), nr_four1(), nr_gammln(), nr_gammq(), nr_gcf(), nr_gser(), nr_median(), nr_realft(), nr_seed(), nr_select(), postprocess_waveform(), print_filter(), process_caltone(), process_diode(), process_dipole(), process_monopole(), process_waveform(), realfft(), rf_addLO(), rf_amplify(), rf_amplify_complex(), rf_mixer(), rf_phase_shifter(), rf_rectify(), setup_calibration(), wfstat_print(), wfstat_reset(), and zplane_transform().

6.5.2.2 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 9 of file bpm_warning.c.

References libbpm_evtnum.

Referenced by complexfft(), complexwf_add(), complexwf_delete(), complexwf_divide(), complexwf_-getamp(), complexwf_getimag(), complexwf_getphase(), complexwf_getreal(), complexwf_multiply(), complexwf_setimag(), complexwf_setreal(), complexwf_subtract(), create_filter(), doublewf_add(), doublewf_basic_stats(), doublewf_delete(), doublewf_divide(), doublewf_multiply(), doublewf_subtract(), get_IQ(), get_mode_amplitude(), get_t0(), intwf_add(), intwf_delete(), intwf_divide(), intwf_multiply(), intwf_subtract(), nr_gcf(), nr_gser(), process_caltone(), process_waveform(), and realfft().

6.6 Numerical routines

6.6.1 Detailed Description

Files

- file **bpm_nr.h**

  libbpm numerical helper routines

- file **dround.c**
- 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_quadinterpol.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**
- file **nr_sinc.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 struct 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_abs2** (**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)
- EXTERN double **nr_quadinterpol** (double x, double x1, double x2, double x3, double y1, double y2, double y3)
- EXTERN double **sinc** (double x)
- EXTERN double **lanczos** (double x, int a)
- EXTERN double **dround** (double x)

Variables

- EXTERN long **bpm_rseed**

6.6.2 Define Documentation

6.6.2.1 #define GCF_ITMAX

Definition at line 30 of file bpm_nr.h.

Referenced by nr_gcf().

6.6.2.2 #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.

6.6.2.3 #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.

6.6.2.4 #define LM_DIF_WORKSZ(npar, nmeas)

see LM_DER_WORKSZ

Definition at line 75 of file bpm_nr.h.

6.6.2.5 #define NR_FFTFORWARD

Perform forward FFT in nr_four

Definition at line 86 of file bpm_nr.h.

6.6.2.6 #define NR_FFTBACKWARD

Perform backward FFT in nr_four

Definition at line 87 of file bpm_nr.h.

6.6.2.7 #define __LM_MEDIAN3(a, b, c)

find the median of 3 numbers

Definition at line 90 of file bpm_nr.h.

6.6.3 Function Documentation

6.6.3.1 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().

6.6.3.2 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().

6.6.3.3 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_ITMAX, and nr_gammln().

Referenced by nr_gammq().

6.6.3.4 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(), and nr_gammln().

Referenced by nr_gammq().

6.6.3.5 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 get_t0().

6.6.3.6 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().

6.6.3.7 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 nr_realft().

6.6.3.8 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().

6.6.3.9 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.

Referenced by nr_rangauss(), and nr_ranuniform().

6.6.3.10 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.

6.6.3.11 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 complexwf_add_noise(), and rf_addLO().

6.6.3.12 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 complexwf_add_ampnoise(), complexwf_add_cwtone(), complexwf_add_dcywave(), complexwf_add_noise(), complexwf_add_phasenoise(), digitise(), doublewf_add_ampnoise(), doublewf_-add_cwtone(), doublewf_add_dcywave(), intwf_add_ampnoise(), intwf_add_cwtone(), and intwf_add_-dcywave().

6.6.3.13 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().

6.6.3.14 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().

6.6.3.15 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 bpm_error().

Referenced by gsl_linalg_householder_hm(), and gsl_linalg_householder_hm1().

6.6.3.16 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 bpm_error().

Referenced by gsl_linalg_householder_hm(), gsl_linalg_householder_hm1(), and gsl_linalg_householder_mh().

6.6.3.17 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.

Referenced by gsl_linalg_householder_hm(), gsl_linalg_householder_hm1(), and gsl_linalg_householder_mh().

6.6.3.18 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.

Referenced by ana_get_svd_coeffs(), gsl_linalg_householder_hm(), gsl_linalg_householder_hm1(), and gsl_linalg_householder_mh().

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

Swap two matrix columns

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2004 Gerard Jungman, Brian Gough

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.

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().

6.6.3.20 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 bpm_error().

Referenced by gsl_linalg_householder_transform().

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

The following line is a generalization of return v->data[i]

Definition at line 61 of file gsl_vector.c.

Referenced by ana_get_svd_coeffs(), gsl_linalg_householder_hm(), gsl_linalg_householder_mh(), and gsl_linalg_householder_transform().

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

The following line is a generalization of v->data[i] = x

Definition at line 70 of file gsl_vector.c.

Referenced by ana_get_svd_coeffs(), and gsl_linalg_householder_transform().

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

applies a householder transformation v,tau to matrix m

Definition at line 8 of file gsl_linalg.c.

References gsl_matrix_column(), gsl_matrix_get(), gsl_matrix_set(), gsl_matrix_submatrix(), and gsl_vector_get().

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

applies a householder transformation v,tau to a matrix being build up from the identity matrix, using the first column of A as a householder vector

Definition at line 96 of file gsl_linalg.c.

References gsl_matrix_column(), gsl_matrix_get(), gsl_matrix_set(), and gsl_matrix_submatrix().

6.6.3.25 EXTERN double gsl_linalg_householder_transform (gsl_vector * *v*)

replace v[0:n-1] with a householder vector (v[0:n-1]) and coefficient tau that annihilate v[1:n-1]

Definition at line 285 of file gsl_linalg.c.

References gsl_blas_dnrm2(), gsl_vector_get(), gsl_vector_set(), and gsl_vector_subvector().

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

applies a householder transformation v,tau to matrix m from the right hand side in order to zero out rows

Definition at line 322 of file gsl_linalg.c.

References gsl_matrix_get(), gsl_matrix_set(), gsl_matrix_submatrix(), and gsl_vector_get().

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

Definition at line 8 of file gsl_blas.c.

Referenced by gsl_linalg_householder_transform().

6.6.3.28 EXTERN gsl_block∗ gsl_block_alloc (const size_t *n*)

Definition at line 8 of file gsl_block.c.

References bpm_error().

6.6.3.29 EXTERN double nr_quadinterpol (double *x*, double *x1*, double *x2*, double *x3*, double *y1*, double *y2*, double *y3*)

Parabolic (quadratic) interpolation routine, give 3 points (x1,y1), (x2,y2) and (x3,y3) and a value x which needs to be interpolated. The function returns y, which is the value of a parabola at point x defined by the 3 points given

Definition at line 8 of file nr_quadinterpol.c.

Referenced by doublewf_getvalue().

6.6.3.30 EXTERN double sinc (double *x*)

The normalised sinc(x) function

Definition at line 8 of file nr_sinc.c.

Referenced by doublewf_getvalue(), and lanczos().

6.6.3.31 EXTERN double lanczos (double *x*, int *a*)

The Lanczos kernel

Definition at line 13 of file nr_sinc.c.

References sinc().

Referenced by doublewf_getvalue().

6.6.3.32 EXTERN double dround (double *x*)

Rounds a value to nearest integers, voids the need for -std=c99 in the compilation

Definition at line 6 of file dround.c.

Referenced by gaussian_filter_coeffs(), intwf_add_ampnoise(), intwf_add_cwtone(), intwf_add_-dcywave(), intwf_cast(), intwf_cast_new(), intwf_derive(), intwf_getvalue(), intwf_integrate(), and intwf_-resample().

6.7 RF simulation routines

6.7.1 Detailed Description

Files

- file **bpm_rf.h**

  libbpm rf simulation routines

- file **rf_addLO.c**
- file **rf_amplify.c**
- file **rf_amplify_complex.c**
- file **rf_mixer.c**
- file **rf_phase_shifter.c**
- file **rf_rectify.c**
- file **rf_setup.c**

Functions

- EXTERN int **rf_setup** (int nsamples, double sfreq)
- EXTERN int **rf_rectify** (**doublewf_t** ∗D, **complexwf_t** ∗RF)
- EXTERN int **rf_addLO** (double amp, double lofreq, enum **bpmphase_t** type, double phase, double phasenoise, **doublewf_t** ∗LO)
- EXTERN int **rf_mixer** (**doublewf_t** ∗RF_Re, **doublewf_t** ∗LO, **doublewf_t** ∗IF)
- EXTERN int **rf_amplify** (**doublewf_t** ∗RF, double dB)
- EXTERN int **rf_amplify_complex** (**complexwf_t** ∗RF, double dB)
- EXTERN int **rf_phase_shifter** (**complexwf_t** ∗RF, double rotation)

Variables

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

6.7.2 Function Documentation

6.7.2.1 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.

6.7.2.2 EXTERN int rf_rectify (doublewf_t * *D*, complexwf_t * *RF*)

Rectifies the given waveform assuming a single diode

Parameters:

D the rectified signal

RF the complex waveform to rectify

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Rectifies the given waveform assuming a single diode

Parameters:

D the rectified signal

RF the complex waveform to rectify

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 15 of file rf_rectify.c.

References bpm_error(), complexwf_getreal(), doublewf_t::ns, and doublewf_t::wf.

6.7.2.3 EXTERN int rf_addLO (double *amp*, double *lofreq*, enum bpmphase_t *type*, double *phase*, double *phasenoise*, doublewf_t * *LO*)

Generates an LO waveform

Parameters:

amp amplitude of the LO signal in Volts

lofreq LO frequency locked or freerunning oscillator phase of the signal, ignored if type is not "locked" phase noise to be added to the waveform

LO generated waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Generates an LO waveform

Parameters:

amp amplitude of the LO signal in Volts

lofreq LO frequency locked or freerunning oscillator phase of the signal, ignored if type is not "locked" phase noise to be added to the waveform

LO generated waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 21 of file rf_addLO.c.

References bpm_error(), doublewf_add_cwtone(), locked, and nr_ranuniform().

6.7.2.4 EXTERN int rf_mixer (doublewf_t * *RF*, doublewf_t * *LO*, doublewf_t * *IF*)

Simulates an ideal mixer

Parameters:

RF signal to mix

LO local oscillator signal to mix with

IF resulting signal containing the up and down converted terms

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Simulates an ideal mixer

Parameters:

RF signal to mix

LO local oscillator signal to mix with

IF resulting signal containing the up and down converted terms

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 17 of file rf_mixer.c.

References bpm_error(), doublewf_copy(), and doublewf_multiply().

6.7.2.5 EXTERN int rf_amplify (doublewf_t * *RF*, double *dB*)

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

$$gain = \sqrt{10^{\frac{db}{20}}}$$

Parameters:

RF waveform to be processed

dB gain (or attenuation) in dB

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

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

$$gain = \sqrt{10^{\frac{db}{20}}}$$

Parameters:

RF waveform to be processed

dB gain (or attenuation) in dB

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 17 of file rf_amplify.c.

References bpm_error(), and doublewf_scale().

6.7.2.6 EXTERN int rf_amplify_complex (complexwf_t * *RF*, double *dB*)

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

$$gain = \sqrt{10^{\frac{db}{20}}}$$

Parameters:

RF waveform to be processed

dB gain (or attenuation) in dB

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

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

$$gain = \sqrt{10^{\frac{db}{20}}}$$

Parameters:

RF waveform to be processed

dB gain (or attenuation) in dB

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 17 of file rf_amplify_complex.c.

References bpm_error(), complexwf_scale(), complex_t::im, and complex_t::re.

6.7.2.7 EXTERN int rf_phase_shifter (complexwf_t * *RF*, double *rotation*)

Rotates the phase of the signal by the amount specified

Parameters:

RF waveform to be processed

rotation phase rotation in degrees

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Rotates the phase of the signal by the amount specified

Parameters:

RF waveform to be processed

rotation phase rotation in degrees

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 16 of file rf_phase_shifter.c.

References bpm_error(), complexwf_scale(), complex_t::im, and complex_t::re.

6.7.3 Variable Documentation

6.7.3.1 EXTERN int rf_nsamples

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

Definition at line 63 of file bpm_rf.h.

Referenced by rf_setup().

6.7.3.2 EXTERN double rf_samplefreq

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

Definition at line 69 of file bpm_rf.h.

Referenced by rf_setup().

6.8 BPM signal simulation routines

6.8.1 Detailed Description

Files

- file **add_mode_response.c**
- file **bpm_simulation.h**

  libbpm waveform simulation routines

- file **digitise.c**
- file **generate_bpmsignal.c**
- file **generate_diodesignal.c**
- file **get_mode_amplitude.c**
- file **get_mode_response.c**
- file **set_temp.c**
- file **set_time.c**

Defines

- #define **K_SAMPLE**
- #define **MODE_DECAY**
- #define **MODE_MAX_SAMPLES**

Functions

- EXTERN int **set_temp** (double TK)
- EXTERN int **set_time** (double ts)
- EXTERN int **generate_bpmsignal** (**bpmconf_t** ∗bpm, **bpmmode_t** ∗mode, **beamconf_t** ∗beam, **doublewf_t** ∗rf)
- EXTERN int **add_mode_response** (**bpmconf_t** ∗bpm, **bpmmode_t** ∗mode, **bunchconf_t** ∗bunch, **doublewf_t** ∗rf)
- EXTERN **complex_t get_mode_amplitude** (**bpmconf_t** ∗bpm, **bpmmode_t** ∗mode, **bunchconf_t** ∗bunch)
- EXTERN **doublewf_t** ∗ **generate_diodesignal** (**doublewf_t** ∗rf, double sens, **filter_t** ∗filt, triggertype_t diode)
- EXTERN int **get_mode_response** (**bpmmode_t** ∗mode)
- EXTERN int **digitise** (**doublewf_t** ∗IF, int nbits, double range_min, double range_max, double clock_jitter, double digi_noise, unsigned int ipmode, **intwf_t** ∗wf)

Variables

- EXTERN double **ambient_temp**
- EXTERN double **system_time**

6.8.2 Define Documentation

6.8.2.1 #define K_SAMPLE

Definition at line 48 of file bpm_simulation.h.

6.8.3 Function Documentation

6.8.3.1 EXTERN int set_temp (double *TK*)

Set ambient temperature

Sets up the ambient temperature

Parameters:

TK ambient temperature in Kelvin

Returns:

BPM_SUCCESS

Definition at line 17 of file set_temp.c.

References ambient_temp.

6.8.3.2 EXTERN int set_time (double *ts*)

Set system time

Sets up the system clock

Parameters:

ts current time in seconds

Returns:

BPM_SUCCESS

Definition at line 17 of file set_time.c.

References system_time.

6.8.3.3 EXTERN int generate_bpmsignal (bpmconf_t ∗ *bpm*, bpmmode_t ∗ *mode*, beamconf_t ∗ *beam*, doublewf_t ∗ *rf*)

Calculates the multi-mode response of a cavity BPM defined using bpminterface structures for a beam containing both one or multiple bunches.

Parameters:

bpm a pointer to the structure defining the bpm

beam a pointer to the structure defining the beam

rf a pointer to were to store the generated waveform

Returns:

BPM_SUCCES upon succes, BPM_FAILURE upon failure

Definition at line 9 of file generate_bpmsignal.c.

References add_mode_response(), bunchconf::arrival_time, bpm_error(), bpmmode::buffer, beamconf::bunch, doublewf(), doublewf_getvalue(), doublewf_reset(), doublewf_t::fs, complexwf_t::fs, bpmmode::name, beamconf::nbunches, doublewf_t::ns, complexwf_t::ns, bpmmode::response, doublewf_t::wf, and WF_QUADRATIC.

6.8.3.4 EXTERN int add_mode_response (bpmconf_t ∗ *bpm*, bpmmode_t ∗ *mode*, bunchconf_t ∗ *bunch*, doublewf_t ∗ *rf*)

Adds the response of a single mode generated by one bunch to the waveform rf, starting at the first sample

Parameters:

bpm a pointer to the structure defining the bpm

mode a pointer to the structure defining a cavity mode

bunch a pointer to the structure defining the current bunch

rf a pointer the waveform the response will be added to

Returns:

BPM_SUCCES upon succes, BPM_FAILURE upon failure

Definition at line 10 of file add_mode_response.c.

References bpm_error(), get_mode_amplitude(), complex_t::im, complexwf_t::ns, doublewf_t::ns, bpmmode::order, complex_t::re, bpmmode::response, complexwf_t::wf, and doublewf_t::wf.

Referenced by generate_bpmsignal().

6.8.3.5 EXTERN complex_t get_mode_amplitude (bpmconf_t * *bpm*, bpmmode_t * *mode*, bunchconf_t * *bunch*)

Returns the complex amplitude of the mode response. The imaginary part is only used when the incline or tilt signal is calculated which has a 90 deg phase offset.

Parameters:

bpm a pointer to the structure defining the bpm

mode a pointer to the structure defining a cavity mode

bunch a pointer to the structure defining the current bunch

Returns:

BPM_SUCCES upon succes, BPM_FAILURE upon failure

Definition at line 9 of file get_mode_amplitude.c.

References bpm_warning(), bunchconf::bpmposition, bunchconf::bpmslope, bpmconf::cav_length, bunchconf::charge, bpmmode::frequency, horiz, complex_t::im, bunchconf::length, bpmmode::order, bpmmode::polarisation, complex_t::re, and bpmmode::sensitivity.

Referenced by add_mode_response().

6.8.3.6 EXTERN doublewf_t* generate_diodesignal (doublewf_t * *rf*, double *sens*, filter_t * *filt*, triggertype_t *diode*)

Rectifies the rf waveform (from the reference cavity) to get a trigger pulse.

Parameters:

rf input waveform

sens diode sensitivity in mV/uW

filt pointer to a filter to apply on the signal

diode type of the diode (pos/neg/bipolar)

dc_out rectified signal

Returns:

a pointer to the generated rectified waveform

Definition at line 11 of file generate_diodesignal.c.

References apply_filter(), bipolar, bpm_error(), doublewf(), m33::e, doublewf_t::fs, negative, doublewf_-t::ns, positive, and doublewf_t::wf.

6.8.3.7 EXTERN int get_mode_response (bpmmode_t * *mode*)

Calculates the normalized complex mode response, the imaginary part is only used to store incline/tilt signals

Parameters:

mode structure containing describing the mode and response buffer

Returns:

BPM_SUCCESS upon success or BPM_FAILURE upon failure

Definition at line 11 of file get_mode_response.c.

References apply_filter(), BANDPASS, bpm_error(), complexwf_reset(), complexwf_setimag(), complexwf_setreal(), create_filter(), delete_filter(), doublewf(), doublewf_delete(), doublewf_integrate(), doublewf_scale(), bpmmode::frequency, complexwf_t::fs, complexwf_t::ns, bpmmode::order, bpm-mode::Q, RESONATOR, bpmmode::response, and doublewf_t::wf.

6.8.3.8 EXTERN int digitise (doublewf_t * *IF*, int *nbits*, double *range_min*, double *range_max*, double *clock_jitter*, double *digi_noise*, unsigned int *ipmode*, intwf_t * *wf*)

Digitises the waveform using the sampling frequency and the number of samples set in the resulting wave-form

Parameters:

IF input waveform to digitse

nbits bit resolution of the ADC

range_min the minimum voltage and

range_max the maximum voltage the ADC can process

clock_jitter ADC clock jitter

digi_noise rms digitiser noise in ADC channels

ipmode interpolation mode for **doublewf_getvalue()** (p. 98)

wf sampled waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 11 of file digitise.c.

References bpm_error(), doublewf_getvalue(), doublewf_t::fs, intwf_t::fs, intwf_add_ampnoise(), nr_-rangauss(), doublewf_t::ns, intwf_t::ns, and intwf_t::wf.

6.8.4 Variable Documentation

6.8.4.1 EXTERN double ambient_temp

Ambient temperature in K

Definition at line 67 of file bpm_simulation.h.

Referenced by set_temp().

6.8.4.2 EXTERN double system_time

Current system time in s

Definition at line 77 of file bpm_simulation.h.

Referenced by set_time().

6.9 Digital Signal Processing Routines

6.9.1 Detailed Description

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

6.9.2 The digital filtering routines

6.9.2.1 General usage Setup a filter using the **create_filter()** (p. 58) routine.

```
filter_t *filter = create_filter( "the_filter", RESONATOR | , 0,
                                  nsamples, 40.*kHz, 8.*kHz, 0., 200. );
```

The arguments the filter expects is a name for the filter (just for esthetic purposes when printing the filter), the filter options, which are explained below, the order of the filter, where it is meaning full (e.g. Butterworth, Bessel, Chebyshev). Then it needs the number of samples in the waveforms which will be filtered by this filter, the sampling frequency and one (optionally two) frequency parameter. For lowpass/highpass filters and the resonater, only the first frequency defines respectively the -3dB frequency level for the low/high pass and the resonance frequency for the resonator (the witdh is defined by the Q value in this case). For bandpass/stop filters the two frequencies are required and define the -3dB level which defines the bandwidth of the filter, with f1 being the lower end frequency and f2 the higher end.

The implemented filters are :

- BESSEL : Bessel IIR filter
- BUTTERWORTH : Butterwordth IIR filter
- CHEBYSHEV : Chebyshev IIR filter
- RESONATOR : Resonators
- GAUSSIAN : Non-causal Gaussian FIR filter

The IIR Bessel, Butterworth and Chebyshev filters can be normalised as lowpass (option LOWPASS) which is the default, highpass (option HIGHPASS), bandstop (option BANDSTOP) or bandpass (option BANDPASS) filters. They are designed with poles and zeros in the s plane that are transformed to the z plane either by bilinear z transform (option BILINEAR_Z_TRANSFORM) or matched z transform (option MATCHED_Z_TRANSFORM). Just "OR" the options together to setup the filter, e.g. :

```
filter_t *filter = create_filter( "lp", BESSEL | HIGHPASS | MATCHED_Z_TRANSFORM, 0,
                                  ns, 40.*kHz, 8.*kHz, 0., 200. );
```

The resonators are designed directly with their 2 poles and 2 zeros in the z plane and can be normalised either as BANDPASS (default), BANDSTOP (or NOTCH) or ALLPASS resonators.

The last argument to the **create_filter()** (p. 58) routine is a parameter which can optionally be given to the filter. It depends on the filter chosen, currently the parameter has meaning for the following filters :

- BESSEL : the parameter defines the ripple in dB, has to be negative !
- RESONATOR : the parameter gives the Q value of the resonator, if you want to have a pure oscillator (so infinite Q), then set the parameter to a negative number or zero.
- GAUSSIAN : the filter cut-off parameter, or the fraction of the gaussian convolution function below which it is set to 0. (default is 0.001)

The filter coefficients for the difference equation are calculated and checked for consistency, upon which they are stored in the filter structure. Once this is done and the filter is setup, application to various waveforms is fairly straightforward. Note that you only have to define your filter once during initialisation. Once setup, it can be used to filter any number of waveforms of the same type.

```
apply_filter( filter, wave );
```

To get an impulse response from the filter into the secified waveform, where the impulse is given at sample 1000, the following routine is implemented.

```
filter_impulse_response( filter, wave, 1000 );
```

This routine creates an impulse function (zero everywhere, except at the sample you enter, where it's value is 1) and puts it through the filter. The FFT of this impulse response gives you the filter characteristic in frequency domain. Also you can check the filter's response to a step function, it's so-called step response :

```
filter_step_response( filter, wave, 1000 )
```

The step response is defined as the response of the filter to an input function which is zero at the beginning and 1 for samples >= the sample you specify.

6.9.2.2 The Bessel, Butterworth and Chebyshev filters

6.9.2.3 The Resonator filter

6.9.2.4 The gaussian filter The gaussian filter is implemented as a FIR convolution with both causal and anti-causal coefficients. Note that the frequency given is treated as the -3dB level for the gaussian. There is an option to restore the definition for bandwith which was used in early ESA processing, being the gaussian sigma, use GAUSSIAN_SIGMA_BW.

6.9.3 The Digital Downconversion Algorithm (DDC)

The digital downconversion routine was developped to process digitised BPM waveforms and to retreive their position and amplitude. It basically implements an RF mixer in software. You need to supply it with the **doublewf_t** (p. 141) holding the waveform to mix down and the frequency for the software LO. Also you need to give a pointer to a low-pass filter in order to filter out the resulting double frequency component from the downmixing. The routine

```
int ddc( doublewf_t *w, double f, filter_t *filter, complexwf_t *dcw );
```

returns then the complex DC waveform (dcw), where it's amplitude and phase can then be used in further calculations for beam position and slope in the BPM. We recommend the usage of a GAUSSIAN low-pass filter for the double frequency filtering as this shows the best phase behaviour combined with linearity (see **create_filter()** (p. 58)).

For fast execution, the DDC routine comes with a buffer which it only allocates once by doing

```
ddc_initialise();
```

This buffer is used in the filtering routine, you can clean up after the execution of the buffer by having

```
ddc_cleanup();
```

6.9.4 Discrete (Fast) Fourier Transforms

The FFT routines in the dsp section of libbpm are based upon the General Purpose FFT Package by Takuya OOURA, 1996-2001, see `http://www.kurims.kyoto-u.ac.jp/~ooura/fft.html` More specifically on it's split-radix fast version (fftsg). These set of routines needs a buffer for bitswapping an a buffer to store a table with sin and cos values so they needn't be calculated for every FFT. The routine

```
fft_initialise( int ns )
```

intialises the buffers for waveforms of a certain sample length ns. Note that ns has to be a power of 2. You can clear the FFT buffers by issuing

```
fft_cleanup( );
```

Then two wrapper routines are implemented which take **doublewf_t** (p. 141) and **complexwf_t** (p. 140) data.

6.9.4.1 Complex Discrete Fourier Transform

The first one is

```
int complexfft( complexwf_t *z, int fft_mode );
```

which takes a complex waveform and performs an FFT in place. The fft_mode argument can be either

- FFT_FORWARD : forward discrete Fourier transform (plus-sign)

$$X[k] = \sum_{j=0}^{n-1} x[j] * \exp(2 * \pi * i * j * k/n), 0 <= k < n$$

- FFT_BACKWARD : backward discrete Fourier transform (minus-sign)

$$X[k] = \sum_{j=0}^{n-1} x[j] * \exp(-2 * \pi * i * j * k/n), 0 <= k < n$$

Note the backward and forward FFT's have a factor of n inbetween them, so to get the orginal wf back after applying both the backward and the forward FFT, you need to divdide by the number of samples z->n.

6.9.4.2 Real Discrete Fourier Transform

The second routine implements the real discrete Fourier transform when having FFT_FORWARD and the other way around when having FFT_BACKWARD.

```
int realfft( doublewf_t *y, int fft_mode, complexwf_t *z );
```

So for FFT_FORWARD

$$Re(X[k]) = \sum_{j=0}^{n-1} a[j] * \cos(2 * \pi * j * k/n), 0 <= k <= n/2$$

$$Im(X[k]) = \sum_{j=0}^{n-1} a[j] * \sin(2 * \pi * j * k/n), 0 < k < n/2$$

and FFT_BACKWARD takes the input frmo the first half (n/2) of the **complexwf_t** (p. 140) and FFTs it, expanding to a **doublewf_t** (p. 141) of length n.

$$X[k] = \frac{(Re(x[0]) + Re(x[n/2]) * \cos(\pi * k))}{2} + \sum_{j=1}^{n/2-1} Re(x[j]) * \cos(2*\pi*j*k/n) + \sum_{j=1}^{n/2-1} Im(x[j]) * \sin(2*\pi*j*k/n), 0 <=$$

6.9.4.3 Reference for FFT routines

- Masatake MORI, Makoto NATORI, Tatuo TORII: Suchikeisan, Iwanamikouzajyouhoukagaku18, Iwanami, 1982 (Japanese)
- Henri J. Nussbaumer: Fast Fourier Transform and Convolution Algorithms, Springer Verlag, 1982
- C. S. Burrus, Notes on the FFT (with large FFT paper list) `http://www-dsp.rice.edu/research/fft/fftnote.asc`

6.9.4.4 Copyright statement for FFT routines

Copyright(C) 1996-2001 Takuya OOURA email: `ooura@mmm.t.u-tokyo.ac.jp` download: `http://momonga.t.u-tokyo.ac.jp/~ooura/fft.html` You may use, copy, modify this code for any purpose and without fee. You may distribute this ORIGINAL package.

6.9.5 DSP example program

There is an example program, which can be found in the examples directory under dsp. It shows how to work with the filtering and the DDC routines...

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include <iostream>

#include <TROOT.h>
#include <TFile.h>
#include <TTree.h>

#include <bpm/bpm_process.h>
#include <bpm/bpm_units.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_nr.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_dsp.h>
#include <bpm/bpm_wf.h>

using namespace std;

int main( int argc, char **argv ) {

  cout << "Welcome to the libbpm DSP sandbox" << endl;

  int ns    = 256;
  double fs = 119.*MHz;

  doublewf_t *w = doublewf( ns, fs );
  doublewf_t *s = doublewf_sample_series( ns, fs );

  doublewf_t *ddc_amp   = doublewf( ns, fs );
  doublewf_t *ddc_phase = doublewf( ns, fs );

  // setup the root trees...
  TFile *rootfile = new TFile( "dsp.root", "recreate" );
  TTree *roottree = new TTree( "dsp", "libbpm dsp tests" );

  int evt;
  double amp, phase;
  double gen_amp, gen_phase;

  // setup the branches in the tree
  roottree->Branch( "evt",       &evt,          "evt/I"             );
  roottree->Branch( "wf",        w->wf,         "wf[256]/D"         );
  roottree->Branch( "s",         s->wf,         "s[256]/D"          );
  roottree->Branch( "gen_amp",   &gen_amp,      "gen_amp/D"         );
  roottree->Branch( "gen_phase", &gen_phase,    "gen_phase/D"       );
  roottree->Branch( "ddc_amp",   ddc_amp->wf,   "ddc_amp[256]/D"    );
  roottree->Branch( "ddc_phase", ddc_phase->wf, "ddc_phase[256]/D" );

  complexwf_t *ddcwf = complexwf( ns, fs );

  filter_t *gauss  = create_filter( "gauss", GAUSSIAN,0,ns,fs,6.*MHz,0.,0.001);
  filter_t *butter = create_filter( "butter", BUTTERWORTH | LOWPASS,4,ns,fs,6.*MHz,0.,0.);
  filter_t *bessel = create_filter( "bessel", BESSEL | LOWPASS,4,ns, fs,6.*MHz,0., 0.);
  filter_t *cheby  = create_filter( "cheby",  CHEBYSHEV | LOWPASS,4,ns,fs,6.*MHz,0.,-10.);

  // init the DDC
  ddc_initialise( ns, fs );

  for ( evt = 1; evt<=1000; evt++ ) {

    // Make the waveform
    gen_amp   = (double) evt * 10.;
    gen_phase = PI / (double) evt;
```

```
    // reset the w to 0... quite important :D
    doublewf_reset( w );

    doublewf_add_dcywave( w, gen_amp, gen_phase, 21.4*MHz, 0.15*usec, 0.2*usec, 0. );

    // do the DDC :)
    if ( ddc( w, 21.4*MHz, gauss, ddcwf ) ) return 1;

    // want to try differen filters ?
    //if ( ddc( w, 21.4*MHz, butter, ddcwf ) ) return 1;
    //if ( ddc( w, 21.4*MHz, bessel, ddcwf ) ) return 1;
    //if ( ddc( w, 21.4*MHz, cheby, ddcwf ) ) return 1;


    // get amplitude and phase from complex wf
    complexwf_getamp( ddc_amp, ddcwf );
    complexwf_getphase( ddc_phase, ddcwf );

    // fill the tree...
    roottree->Fill();

    if ( evt % 100 == 0 ) cout << "Simulated " << evt << " events." << endl;
  }

  // clear the DDC memory buffers
  ddc_cleanup();

  rootfile->Write();
  rootfile->Close();

  delete_filter( gauss );
  delete_filter( butter );
  delete_filter( bessel);
  delete_filter( cheby );

  complexwf_delete( ddcwf );

  doublewf_delete( w );
  doublewf_delete( s );
  doublewf_delete( ddc_amp );
  doublewf_delete( ddc_phase );

  return 0;
}
```

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 **ddc.c**
- file **delete_filter.c**
- file **discrete_fourier_transforms.c**
- file **filter_impulse_response.c**
- file **filter_step_response.c**
- file **gaussian_filter_coeffs.c**
- file **norm_phase.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 **CAUSAL**
- #define **ANTICAUSAL**
- #define **NONCAUSAL**
- #define **GAUSSIAN_SIGMA_BW**
- #define **LOWPASS**
- #define **HIGHPASS**
- #define **BANDPASS**
- #define **BANDSTOP**
- #define **NOTCH**
- #define **ALLPASS**
- #define **FIR**
- #define **IIR**
- #define **MAXORDER**
- #define **MAXPZ**
- #define **FILT_EPS**
- #define **MAX_RESONATOR_ITER**
- #define **FFT_FORWARD**
- #define **FFT_BACKWARD**

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, **doublewf_t** ∗w)
- EXTERN void **print_filter** (FILE ∗of, **filter_t** ∗f)
- EXTERN void **delete_filter** (**filter_t** ∗f)
- EXTERN int **filter_step_response** (**filter_t** ∗f, **doublewf_t** ∗w, int itrig)
- EXTERN int **filter_impulse_response** (**filter_t** ∗f, **doublewf_t** ∗w, 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 **gaussian_filter_coeffs** (**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)
- EXTERN int **ddc_initialise** (int ns, double fs)
- EXTERN void **ddc_cleanup** (void)
- int **ddc** (**doublewf_t** ∗w, double f, **filter_t** ∗filter, **complexwf_t** ∗dcw, **doublewf_t** ∗bufre, **doublewf_t** ∗bufim)
- EXTERN int **fft_gen_tables** (void)
- EXTERN int **fft_initialise** (int ns)
- EXTERN void **fft_cleanup** (void)
- EXTERN int **complexfft** (**complexwf_t** ∗z, int fft_mode)
- EXTERN int **realfft** (**doublewf_t** ∗y, int fft_mode, **complexwf_t** ∗z)
- EXTERN void **norm_phase** (double ∗phase)

6.9.6 Define Documentation

6.9.6.1 #define BESSEL

Bitmask for Bessel filter

Definition at line 384 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

6.9.6.2 #define BUTTERWORTH

Bitmask for Butterworth filter

Definition at line 385 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

6.9.6.3 #define CHEBYSHEV

Bitmask for Chebyshev filter

Definition at line 386 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

6.9.6.4 #define RAISEDCOSINE

Bitmask for Raised Cosine filter

Definition at line 387 of file bpm_dsp.h.

6.9.6.5 #define RESONATOR

Bitmask for Resonator filter

Definition at line 388 of file bpm_dsp.h.

Referenced by create_filter(), and get_mode_response().

6.9.6.6 #define GAUSSIAN

Bitmask for Gaussian filter

Definition at line 389 of file bpm_dsp.h.

Referenced by create_filter().

6.9.6.7 #define BILINEAR_Z_TRANSFORM

Get z poles via bilinear z transform from s plane

Definition at line 391 of file bpm_dsp.h.

6.9.6.8 #define MATCHED_Z_TRANSFORM

Get z poles via matches z transform from s plane

Definition at line 392 of file bpm_dsp.h.

Referenced by zplane_transform().

6.9.6.9 #define NO_PREWARP

Don't do the prewarp correction

Definition at line 393 of file bpm_dsp.h.

Referenced by create_filter().

6.9.6.10 #define CAUSAL

Filter is purely causal (only depends on past)

Definition at line 394 of file bpm_dsp.h.

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

6.9.6.11 #define ANTICAUSAL

.... purely anticausal (only depends on future)

Definition at line 395 of file bpm_dsp.h.

Referenced by apply_filter(), and print_filter().

6.9.6.12 #define NONCAUSAL

Filter is both causal and acausal

Definition at line 396 of file bpm_dsp.h.

Referenced by create_filter().

6.9.6.13 #define GAUSSIAN_SIGMA_BW

Gaussian sigma bandwidth in stead of -3 dB (def)

Definition at line 397 of file bpm_dsp.h.

Referenced by gaussian_filter_coeffs().

6.9.6.14 #define LOWPASS

Normalise filter as lowpass

Definition at line 399 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and normalise_filter().

6.9.6.15 #define HIGHPASS

Normalise filter as highpass

Definition at line 400 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and normalise_filter().

6.9.6.16 #define BANDPASS

Normalise filter as bandpass

Definition at line 401 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), get_mode_response(), and normalise_filter().

6.9.6.17 #define BANDSTOP

Normalise filter as bandstop

Definition at line 402 of file bpm_dsp.h.

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

6.9.6.18 #define NOTCH

Normalise filter as notch filter (=bandstop)

Definition at line 403 of file bpm_dsp.h.

6.9.6.19 #define ALLPASS

Normalise filter as allpass (resonator)

Definition at line 404 of file bpm_dsp.h.

Referenced by create_resonator_representation().

6.9.6.20 #define FIR

Filter is of FIR type

Definition at line 406 of file bpm_dsp.h.

Referenced by apply_filter(), and create_filter().

6.9.6.21 #define IIR

Filter is of IIR type

Definition at line 407 of file bpm_dsp.h.

Referenced by create_filter().

6.9.6.22 #define MAXORDER

Maximum filter order

Definition at line 409 of file bpm_dsp.h.

6.9.6.23 #define MAXPZ

Maximum number of poles and zeros >2∗MAXORDER

Definition at line 410 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), create_resonator_representation(), and gaussian_filter_-coeffs().

6.9.6.24 #define FILT_EPS

A small number used in bpmdsp

Definition at line 411 of file bpm_dsp.h.

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

6.9.6.25 #define MAX_RESONATOR_ITER

Maximum iterations in resonator poles calculation

Definition at line 412 of file bpm_dsp.h.

Referenced by create_resonator_representation().

6.9.6.26 #define FFT_FORWARD

Perform FFT from time -> frequency

Definition at line 414 of file bpm_dsp.h.

Referenced by complexfft(), fft_waveform(), and realfft().

6.9.6.27 #define FFT_BACKWARD

Perform FFT from frequency -> time

Definition at line 415 of file bpm_dsp.h.

Referenced by complexfft(), and realfft().

6.9.7 Function Documentation

6.9.7.1 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. 60).

Definition at line 10 of file create_filter.c.

References filter_t::alpha1, filter_t::alpha2, BESSEL, bpm_error(), bpm_warning(), BUTTERWORTH, calculate_filter_coefficients(), CAUSAL, filter_t::cheb_ripple, CHEBYSHEV, filter_t::cplane, create_-resonator_representation(), create_splane_representation(), filter_t::f1, filter_t::f2, FIR, filter_t::fs, filter_-t::gauss_cutoff, GAUSSIAN, gaussian_filter_coeffs(), IIR, filter_t::name, NO_PREWARP, NON-CAUSAL, 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().

Referenced by get_mode_response().

6.9.7.2 EXTERN int apply_filter (filter_t $*$ *f*, doublewf_t $*$ *w*)

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 ANTICAUSAL, bpm_error(), CAUSAL, FIR, filter_t::gain, filter_t::ns, filter_t::nxc, filter_-t::nxc_ac, filter_t::nyc, filter_t::options, doublewf_t::wf, filter_t::wfbuffer, filter_t::xc, filter_t::xc_ac, filter_t::xv, filter_t::xv_ac, filter_t::yc, filter_t::yv, and filter_t::yv_ac.

Referenced by ddc(), filter_impulse_response(), filter_step_response(), generate_diodesignal(), and get_-mode_response().

6.9.7.3 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 ANTICAUSAL, bpm_error(), CAUSAL, filter_t::cplane, filter_t::dc_gain, filter_t::fc_gain, FILT_EPS, filter_t::gain, filter_t::hf_gain, filter_t::name, filter_t::nxc, filter_t::nxc_ac, filter_t::nyc, filter_t::options, print_filter_representation(), filter_t::xc, filter_t::xc_ac, and filter_t::yc.

6.9.7.4 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 7 of file delete_filter.c.

References filter_t::cplane, and filter_t::wfbuffer.

Referenced by get_mode_response().

6.9.7.5 EXTERN int filter_step_response (filter_t * *f*, doublewf_t * *w*, 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(), filter_t::ns, and doublewf_t::wf.

6.9.7.6 EXTERN int filter_impulse_response (filter_t * *f*, doublewf_t * *w*, 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(), filter_t::ns, and doublewf_t::wf.

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

This routine returns a pointer to a filter representation **filterrep_t** (p. 148) 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 BESSEL, bpm_error(), BUTTERWORTH, filter_t::cheb_ripple, CHEBYSHEV, filterrep_-t::npoles, filter_t::options, filter_t::order, and filterrep_t::pole.

Referenced by create_filter().

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

This routine returns a pointer to a filter representation **filterrep_t** (p. 148) 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(), ALLPASS, filter_t::alpha1, BANDSTOP, bpm_error(), 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().

6.9.7.9 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(), MATCHED_Z_TRANSFORM, filterrep_t::npoles, filterrep_t::nzeros, filter_t::options, filterrep_t::pole, and filterrep_t::zero.

Referenced by create_filter().

6.9.7.10 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 filterrep_t::npoles, filterrep_t::nzeros, filterrep_t::pole, and filterrep_t::zero.

Referenced by print_filter().

6.9.7.11 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 BANDPASS, BANDSTOP, bpm_error(), 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().

6.9.7.12 EXTERN int calculate_filter_coefficients (filter_t ∗ *f*)

Calculates the filter coefficients from the z plane representation for Butterworth, Chebyshev, Bessel and Resonators. 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, 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().

6.9.7.13 EXTERN int gaussian_filter_coeffs (filter_t ∗ *f*)

Calculates the gaussian filter coefficients from the original gaussian filter implementation in the digital downconversion algortithm in Yury's code. Note that this filter is implemented as a FIR non-causal filter.

Parameters:

f the filter structure with the coefficients to fill

Returns:

BPM_SUCCESS upon success or BPM_FAILURE upon failure.

Definition at line 8 of file gaussian_filter_coeffs.c.

References bpm_error(), dround(), filter_t::f1, filter_t::fs, filter_t::gain, filter_t::gauss_cutoff, GAUSSIAN_SIGMA_BW, MAXPZ, filter_t::ns, filter_t::nxc, filter_t::nxc_ac, filter_t::options, filter_t::xc, and filter_t::xc_ac.

Referenced by create_filter().

6.9.7.14 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(), and FILT_EPS.

Referenced by calculate_filter_coefficients(), and create_resonator_representation().

6.9.7.15 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. 140))

Definition at line 44 of file calculate_filter_coefficients.c.

Referenced by calculate_filter_coefficients(), and create_resonator_representation().

6.9.7.16 EXTERN int ddc_initialise (int *ns*, double *fs*)

Initialises and allocates memory for the DDC buffers with the correct number of samples and sampling frequency

Parameters:

- ***ns*** Nuber of samples in waveforms to be processed
- ***fs*** The sampling frequency of the waveforms

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 50 of file ddc.c.

References bpm_error(), and doublewf().

6.9.7.17 EXTERN void ddc_cleanup (void)

Clears up and frees the buffer memory for the ddc routines

Definition at line 70 of file ddc.c.

References doublewf_delete().

6.9.7.18 int ddc (doublewf_t ∗ *w*, double *f*, filter_t ∗ *filter*, complexwf_t ∗ *dcw*, doublewf_t ∗ *bufre*, doublewf_t ∗ *bufim*)

Do a digital downconversion on the waveform f. The routine returns a complex DC waveform "wdc". If the buffer arguments are NULL pointers, the DDC routine will use an internal buffer. This is a good option when all the BPMs in the system have the same sampling frequency and number of samples.

Parameters:

w The waveform of doubles to process

f The frequency of the digital local oscillator

filter The lowpass filter to get rid of the 2omega component

dcw The complex DC waveform

bufre The real ddc buffer

bufim The imaginary ddc buffer

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 78 of file ddc.c.

References apply_filter(), complexwf_setimag(), complexwf_setreal(), doublewf_t::fs, complexwf_t::fs, doublewf_t::ns, complexwf_t::ns, and doublewf_t::wf.

Referenced by ddc_waveform().

6.9.7.19 EXTERN int fft_gen_tables (void)

Regenerates the sin/cos tables that are needed for the fast DFT algorithm.

Definition at line 116 of file discrete_fourier_transforms.c.

References bpm_error().

Referenced by fft_initialise().

6.9.7.20 EXTERN int fft_initialise (int *ns*)

This one initialised the FFT buffers, checks whether they are large enough for the given number of samples and frees and re-allocates memory where necessary

Parameters:

ns The number of samples in the waveforms to be transformed

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 130 of file discrete_fourier_transforms.c.

References bpm_error(), and fft_gen_tables().

6.9.7.21 EXTERN void fft_cleanup (void)

This routine frees up the memory used by the FFT buffers

Definition at line 163 of file discrete_fourier_transforms.c.

6.9.7.22 EXTERN int complexfft (complexwf_t ∗ *z*, int *fft_mode*)

Executes a complex fast fourier transform in line. See the reference guide for details.

Parameters:

z The complex waveform to transform (original waveform is destroyed) Note that the number of samples need to be a power of 2.

fft_mode Specifies whether to do the forward or backward transform

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 178 of file discrete_fourier_transforms.c.

References bpm_error(), bpm_warning(), FFT_BACKWARD, FFT_FORWARD, complex_t::im, complexwf_t::ns, complex_t::re, and complexwf_t::wf.

6.9.7.23 EXTERN int realfft (doublewf_t ∗ *y*, int *fft_mode*, complexwf_t ∗ *z*)

Executes a real fast fourier transform, between the real waveform y and the complex waveform z. See documentation for further explanation.

Parameters:

y Pointer to the real wavefrom

fft_mode Specifies whether to do the forward or backward transform

z Pointer to the complex waveform

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 230 of file discrete_fourier_transforms.c.

References bpm_error(), bpm_warning(), FFT_BACKWARD, FFT_FORWARD, complex_t::im, complexwf_t::ns, complex_t::re, complexwf_t::wf, and doublewf_t::wf.

Referenced by fft_waveform().

6.9.7.24 EXTERN void norm_phase (double * *phase*)

Normalises the phase, to the interval [0,2pi[

Parameters:

phase Pointer to the phase value to normalise

Definition at line 8 of file norm_phase.c.

Referenced by complexwf_getphase(), complexwf_getphase_new(), postprocess_waveform(), process_-caltone(), and process_waveform().

6.10 BPM Processing Routines

6.10.1 Detailed Description

This set of routines contains the BPM digitised waveform processing routines to go from a sis digitised waveform to position and slope information.

6.10.2 General structure of the BPM signal processing

The BPM signal processing algorithms are centered around a few top-level routines which need to called by a standard user. All make use of a number of BPM data structures which hold BPM configuration data (bpmconf_t), processed BPM information (bpmproc_t) or BPM calibration information (bpmcalib_t). As the BPM processing algorithms make extensive use of the bpmdsp module, the BPM signals need to be encapsulated in a **doublewf_t** (p. 141) waveform before feeding them to these processing routines. The top-level processing routines have a mode bitword which provides some processing options that the user can feed into the processing algorithm.

6.10.2.1 Diode signal processing Since the idea was to unify the processing into one coherent set of data structures, the diode or trigger information had to be fitted into the same framework as the BPM data. This is the function call :

```
int process_diode( doublewf_t *signal, bpmconf_t *conf, bpmproc_t *proc );
```

So the diode pulse has to be fitted into a **doublewf_t** (p. 141) along with a bpmconf_t structure conf. The routine first checks the flag **bpmconf_t::cav_type** (p. 125) for the cavity type. This should be of type diode for the routine to proceed. It then calls the fit_diodepulse routine onto the signal, which returns the fitted t0 into the bpmproc_t structure as proc->t0.

Attention:

Note that there is the possibility to abuse a dipole or monopole signal as a trigger pulse. In this case the process_diode routine will determine the RMS of the noise in front of the digitised dipole/monopole signal (first 20 samples) and return the timestamp in **bpmproc_t::t0** (p. 133) of the first sample which is 10 times largers than this RMS value. For this behaviour, the **bpmconf_t::cav_type** (p. 125) setting is irrelevant but the **bpmconf_t::forced_trigger** (p. 129) value has to be set to 1. Note that this behaviour is normally not needed an for experimental purposes only.

6.10.2.2 Monopole signal processing For monopole cavities one only needs to determine the amplitude and phase, so no post-processing to get to position and slope using a reference cavity and calibration information is needed. Therefore the process_monopole routine is basically a wrapper around the process_-waveform routine which does exactly this determination of the amplitude and phase. The function call is :

```
int process_monopole( doublewf_t *signal, bpmconf_t *bpm, bpmproc_t *proc,
                      bpmproc_t *trig, unsigned int mode );
```

This routine basically is a wrapper around

```
int process_waveform( doublewf_t *signal, bpmconf_t *bpm, bpmproc_t *proc,
                      bpmproc_t *trig, unsigned int mode );
```

and handles all the processing steps flagged by the mode bitword. Chronologically it executes the following steps :

- Check whether the waveform was saturated or not. This is done by a call to check_saturation, which needs the **doublewf_t** (p. 141) signal obviously and the ADC resolution set by the number of bits in **bpmconf_t::digi_nbits** (p. 127). It returns whether the waveform was saturated (saved in **bpmproc_t::saturated** (p. 133)) and assigns the sample number of the first unsaturated sample in the waveform to **bpmproc_t::iunsat** (p. 133).

- Then process_waveform goes on with subtracting the pedestal of the waveform by getting the average and RMS of the first 20 samples in the waveform using get_pedestal and storing the results in **bpmproc_t::voltageoffset** (p. 133) and **bpmproc_t::ampnoise** (p. 133). It subsequently subtracts this voltage offset from each sample in the waveform.

- Then the t0 time is set. If the process_waveform has trigger information available in the form of a bpmproc_t trigger argument wich was handled by process_diode, then the routine will assume this information has to be used as t0 and will copy the trigger->t0 value to it's own **bpmproc_t::t0** (p. 133). If a bpmproc_t trigger argument is not available (NULL pointer), the process_waveform routine will assume the t0 has been set fixed by the BPM configuration (external clocking) and will use and copy the **bpmconf_t::t0** (p. 127) to it's **bpmproc_t::t0** (p. 133) value. The **bpmproc_t::t0** (p. 133) is furtheron used in the rest of the processing as the starting time for this cavity signal.

- If the PROC_DO_FFT flag has been set in the mode bitword, the process_waveform routine will compute the waveform FFT by calling fft_waveform from the bpmdsp module and storing the result in **bpmproc_t::ft** (p. 134). If this is succesfull, the code will go on to check whether this fourier transform needs to be fitted for it's frequency and decay time (Lorentz line width). This is done by calling fit_fft.

  Attention:

  This routine is a little experimental and can easily by replaced by the user with some other package e.g. ROOT. The full complex fourier waveform is available in the **bpmproc_t::ft** (p. 134) as a **complexwf_t** (p. 140).

- If the PROC_DO_FIT flag has been set in the mode bitword, the process_waveform routine will try to fit a decaying sinewave to the waveform, attempting to extract amplitude, phase, frequency and decay time.

  Attention:

  This routine is quite experimental as well and needs proper checking before it can be used stabily ! I recommend using a proper fitting package such as MINUIT to fit the waveforms to a decaying sine wave.

- If the PROC_DO_DDC flag has been set in the mode bitword, the process_waveform routine will perform the digital downconversion on the waveform. As this is a more complex algorithm, we will go into a bit more detail here.
  - First, we have to tell the DDC algoritm where to get it's frequency and decay time from. By default the algorithm will use in both cases the frequency and decay time which are set in the cavities configuration, being **bpmconf_t::ddc_freq** (p. 128) and **bpmconf_t::ddc_tdecay** (p. 128). However, if the flag(s) PROC_DDC_FITFREQ and/or PROC_DDC_FITTDECAY is/are present and the fits (see previous item) were succesfull, the ddc algorithm will use the fitted frequency and decaytime values. Alternatively, if the flag(s) PROC_DDC_FFTFREQ and/or PROC_DDC_FFTTDECAY are/is present, the ddc algoritm will use the frequency and decay time derived from the fitted lorentz lineshape of the waveforms fourier transform.

- Next the DDC algoritm handls the saturation if present (was set by the **bpmproc_t::saturated** (p. 133)) flag already. If the waveform was saturated, we will shift the position of the sample time to the last unsaturated sample.

  Attention:

  Since people haven't converged on a proper way to handle saturation, this is a bit of an open point in the code. At the moment, the ddc_tSample is set to the last unsaturated sample, but one should take into account somehow the bandwidth of the DDC filter, which is not done. I've left it as it is, with the wise advice to store the **bpmproc_t::saturated** (p. 133) flag into the user data and simply cut away those pulses.

  If no saturation is present, the sampling point (expressed in time-units, not sampled) of the DDC algoritm is set to the t0 time (starting point of the waveform) + a constant time offset, which can be tweaked in optimisation.

  ```
  proc->ddc_tSample = proc->t0 + bpm->ddc_tOffset;
  ```

- After the sampling time has been calculated in the previous step, it is converted into a sample number and stored in **bpmproc_t::ddc_iSample** (p. 135).

- Then the real downconversion is done, by default libbpm will try to use the optimised ddc_sample_-waveform routine to save CPU cycles, but if the full DDC is requested by the mode flag PROC_-DDC_FULL, it will go through the entire waveform and convert it to DC using the frequency set as explained previously. The routine that is called is ddc_waveform which basically needs the the pedestal subtraced **doublewf_t** (p. 141) waveform, the frequency of downconversion, a 2 omega filter, defined by a **filter_t** (p. 143) structure having the correct type (lowpass) and bandwidth already define and stored in **bpmconf_t::ddc_filter** (p. 128). The full complex downconverted waveform is stored in the case of full ddc in **bpmproc_t::dc** (p. 134). The amplitude and phase are calculated at the t0 time by extrapolating the phase and amplitude back from the sampling point at **bpmproc_-t::ddc_iSample** (p. 135). The ddc_sample_waveform returns these values directly, but does it internally by extrapolation from the sampling time as well, one therefore needs to provide t0, tdecay and iSample as additional arguments to ddc_sample_waveform compared to ddc_waveform.

- After this is done, the determined phase is normalised in between 0 and 2pi.

6.10.2.3 Dipole signal processing

Dipole cavity waveforms first need to undergo the same processing step as monopole waveforms, to determine their phase and amplitude. After that position and slope information need to be determined using the calibration information. The routine

```
int process_dipole( doublewf_t *signal, bpmconf_t *bpm, bpmcalib_t *cal, bpmproc_t *proc,
                    bpmproc_t *trig, bpmproc_t *ampref, bpmproc_t *phaseref,
                    unsigned int mode );
```

is therefore a wrapper around the following two core routines :

```
int process_waveform( signal, bpm, proc, trig, mode );
int postprocess_waveform( bpm, proc, cal, ampref, phaseref, mode );
```

Attention:

If the PROC_CORR_GAIN (or PROC_CORR_AMP, PROC_CORR_PHASE) flag is set in the mode word, the process_dipole routine will correct the gains based upon the latest calibration tone information stored in the **bpmproc_t::ddc_ct_amp** (p. 136) etc variables and comparing them to the **bpmcalib_t::ddc_ct_amp** (p. 122) at the time of calibration. This is done by a call to correct_gain

The process_waveform is explained under the process_monopole cavity, the postprocess_waveform routine executes the following :

- Firstly the routine calculates the I and Q for the dipole cavity from the amplitude and phase references. This is done by a call to get_IQ, and the values are stored in **bpmproc_t::ddc_Q** (p. 135), **bpmproc_t::ddc_I** (p. 135) for the DDC information and **bpmproc_t::fit_Q** (p. 136), **bpmproc_t::fit_I** (p. 136) for the fitted information.

- For dipole cavities, the real phase information that means anything is the phase difference between the reference cavity and the dipole cavity. This get's stored into **bpmproc_t::ddc_phase** (p. 135) and/or **bpmproc_t::fit_phase** (p. 136). If the flag PROC_RAW_PHASE is set in the mode word, this is skipped.

- Using the I and Q information, the position and slope are calculated

6.10.3 Processing flow

The question now is how to organise the processing flow from the digitised waveform data. Before being able to obtain positions and slopes, the user will need to have processed all the trigger (diode) pulses. And thereafter the monopole waveforms in the event. After that positions and slopes can be calculted using the process_diopole routine. Note that the monopole waveforms depend on the trigger information in the case of internal triggering using a trigger pulse, so a good way to proceed is first to all the trigger pulses, than all the monopole pulses and then all the dipole waveforms.

Alternatively the user can first use the routine process_waveform on all of the waveforms (together with processing the trigger information). After this is done, the user can use the postprocess_waveform routine to perform the post-processing on the dipole waveforms.

6.10.4 About trigger pulses, internal vs. external clock

The SIS ADCs can be triggered by using an external clock in which case all the modules in the system are synchronised and no trigger pulses are needed. Because of the way the processing is setup in process_waveform, the user has to be mindfull of a number of things depending on whether the ADC modules are triggered internally (and a trigger pulse is available) or whether they are triggered externally, synchronised to the beam clock, in which case the starting time (t0) of the pulses should be constant for each individual BPM signal.

6.10.4.1 External clock triggering In this case, the t0 should be set in the BPM configuration under **bpmconf_t::t0** (p. 127). During the processing this value will be used and copied to **bpmproc_t::t0** (p. 133). The bpmconf_t::tOffset defines the offset from this t0 of the pulse of the sampling point in the waveform such that

```
proc->ddc_tSample = proc->t0 + bpm->ddc_tOffset;
```

This mode will be assumed automatically in the absence of the 4th argument of process_waveform (bpmproc_t *trig = NULL).

6.10.4.2 Internal clock triggering There the **bpmconf_t::t0** (p. 127) value is ignored and no t0 value needs to be specified before hand since it will be fitted from the diode/trigger pulse. In this case the 4th argument of process_waveform needs to be present. Also,**the bpmconf_t::tOffset keeps it's definition exaclty the same as in the external clock case**. It is the time difference between the sample time and the starttime of the waveform t0, which in this case got fit instead of being fixed.

6.10.5 calibration tone information

The calibration tone information is kept in two locations. Firstly at the time of calibration, the user should make sure that the latest calibration tone information is set in the bpmcalib_t structure under **bpmcalib_-t::ddc_ct_amp** (p. 122) and **bpmcalib_t::ddc_ct_phase** (p. 122) and analoguous for the parameters for the fitted processing. Than each time a calibration tone pulse is encountered, the user should pass the phase and amplitude of the calibration tone on to the **bpmproc_t::ddc_ct_amp** (p. 136) and **bpmproc_t::ddc_-ct_phase** (p. 136) and therefore always keep the lateste calibration tone information in this location. Each call to

```
int correct_gain( bpmproc_t *proc, bpmcalib_t *cal, unsigned int mode )
```

then corrects the phase and amplitude of the current pulse by scaling the amplitude with the ratio between the caltone amplitude at the time of calibration and the lastest one and shifting the phase by the phase difference between the phase of the calibration tone at the time of BPM calibration and the latest phase recorded in the **bpmproc_t::ddc_ct_phase** (p. 136) variable (or **bpmproc_t::fit_ct_phase** (p. 137)).

Attention:

I've include a mode bitword, which takes the flags PROC_CORR_GAIN to correct both amplitude and phase, and PROC_CORR_AMP, PROC_CORR_PHASE to correct only one parameter individually. This is done since e.g. for internal clocking, when the ADC's are not synchronised to each other, it is not really clear where to sample the waveform unless a trigger is supplied in the ADC. For external synchronized clocking, we can just give a fixed sample number, stored in the bpm configuration under **bpmconf_t::ddc_ct_iSample** (p. 129).

Files

- file **bpm_process.h**

  libbpm main processing routines

- file **check_saturation.c**
- file **correct_gain.c**
- file **ddc_sample_waveform.c**
- file **ddc_waveform.c**
- file **downmix_waveform.c**
- file **fft_waveform.c**
- file **fit_diodepulse.c**
- file **fit_fft.c**
- file **fit_waveform.c**

- file **get_IQ.c**
- file **get_pedestal.c**
- file **get_pos.c**
- file **get_slope.c**
- file **get_t0.c**
- file **postprocess_waveform.c**
- file **process_caltone.c**
- file **process_diode.c**
- file **process_dipole.c**
- file **process_monopole.c**
- file **process_waveform.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_FULL**
- #define **PROC_FIT_DDC**
- #define **PROC_FIT_FFT**
- #define **PROC_RAW_PHASE**
- #define **PROC_CORR_AMP**
- #define **PROC_CORR_PHASE**
- #define **PROC_CORR_GAIN**

Functions

- EXTERN int **process_diode** (**doublewf_t** ∗signal, **bpmconf_t** ∗conf, **bpmproc_t** ∗proc)
- EXTERN int **process_monopole** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmcalib_t** ∗cal, **bpmproc_t** ∗proc, **bpmproc_t** ∗trig, unsigned int mode)
- EXTERN int **process_dipole** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmcalib_t** ∗cal, **bpmproc_t** ∗proc, **bpmproc_t** ∗trig, **bpmproc_t** ∗ampref, **bpmproc_t** ∗phaseref, unsigned int mode)
- EXTERN int **process_waveform** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmproc_t** ∗proc, **bpmproc_t** ∗trig, unsigned int mode)
- EXTERN int **postprocess_waveform** (**bpmconf_t** ∗bpm, **bpmproc_t** ∗proc, **bpmcalib_t** ∗cal, **bpmproc_t** ∗ampref, **bpmproc_t** ∗phaseref, unsigned int mode)
- EXTERN int **process_caltone** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmproc_t** ∗proc, unsigned int mode)
- EXTERN int **correct_gain** (**bpmproc_t** ∗proc, **bpmcalib_t** ∗cal, unsigned int mode)
- EXTERN int **fit_waveform** (**doublewf_t** ∗w, double t0, double i_freq, double i_tdecay, double i_amp, double i_phase, double ∗freq, double ∗tdecay, double ∗amp, double ∗phase)
- EXTERN int **fit_diodepulse** (**doublewf_t** ∗w, double ∗t0)
- EXTERN int **fft_waveform** (**doublewf_t** ∗w, **complexwf_t** ∗ft)

- EXTERN int **fit_fft_prepare** (**complexwf_t** ∗ft, int ∗n1, int ∗n2, double ∗amp, double ∗freq, double ∗fwhm)
- EXTERN int **fit_fft** (**complexwf_t** ∗ft, double ∗freq, double ∗tdecay, double ∗A, double ∗C)
- EXTERN int **check_saturation** (**doublewf_t** ∗w, int nbits, int ∗iunsat)
- EXTERN int **downmix_waveform** (**doublewf_t** ∗w, double frequency, **complexwf_t** ∗out)
- EXTERN int **ddc_waveform** (**doublewf_t** ∗w, double frequency, **filter_t** ∗filt, **complexwf_t** ∗dc, **doublewf_t** ∗buf_re, **doublewf_t** ∗buf_im)
- EXTERN int **ddc_sample_waveform** (**doublewf_t** ∗w, double frequency, **filter_t** ∗filt, int iSample, double t0, double tdecay, double ∗amp, double ∗phase, **doublewf_t** ∗buf_re, **doublewf_t** ∗buf_im)
- EXTERN int **get_pedestal** (**doublewf_t** ∗wf, int range, double ∗offset, double ∗rms)
- EXTERN int **get_t0** (**doublewf_t** ∗w, 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)

6.10.6 Define Documentation

6.10.6.1 #define PROC_DEFAULT

Definition at line 331 of file bpm_process.h.

6.10.7 Function Documentation

6.10.7.1 EXTERN int process_diode (doublewf_t ∗ *signal*, bpmconf_t ∗ *conf*, 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. The routine checks what the signal type (conf->cav_type) is and when it really is a diode pulse, it will fit the pulse and return t0, otherwise (when the signal is a monopole or dipole signal), it will determine the onset of the waveform by looking where the signal's absolute value exceeds 10 ∗ the noise RMS at the beginning of the waveform.

Parameters:

signal The bpm signal

conf The bpm configuration structure

proc The processed trigger structure (containing the t0)

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 9 of file process_diode.c.

References bpm_error(), bpmconf::cav_type, diode, doublewf_basic_stats(), fit_diodepulse(), bpmconf::forced_trigger, doublewf_t::fs, wfstat_t::mean, bpmconf::name, doublewf_t::ns, wfstat_t::rms, bpmproc::t0, and doublewf_t::wf.

6.10.7.2 EXTERN int process_monopole (doublewf_t ∗ *signal*, bpmconf_t ∗ *bpm*, bpmcalib_t ∗ *cal*, bpmproc_t ∗ *proc*, bpmproc_t ∗ *trig*, unsigned int *mode*)

Top-level routine which is basically a wrapper around process_waveform and correct_gain to take into account the calibration tone data. See more in details documentation in those routines.

Parameters:

signal The **doublewf_t** (p. 141) encoded BPM signal

bpm The bpm configuration structure

cal The bpm calibration structure, needed for the gain correction

proc The processed data structure

trig 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 11 of file process_monopole.c.

References bpm_error(), correct_gain(), bpmconf::name, and process_waveform().

6.10.7.3 EXTERN int process_dipole (doublewf_t * *signal*, bpmconf_t * *bpm*, bpmcalib_t * *cal*, bpmproc_t * *proc*, bpmproc_t * *trig*, bpmproc_t * *ampref*, bpmproc_t * *phaseref*, unsigned int *mode*)

Top-level routine which is a wrapper around process_waveform, correct_gain and postprocess_waveform. See more details in the documentation of those individual routines.

Parameters:

signal The **doublewf_t** (p. 141) encoded BPM signal

bpm The bpm configuration structure

cal The bpm calibration structure, needed for the gain correction

proc The processed data structure

trig The structure with processed trigger info for that waveform

ampref The already processed amplitude reference bpmproc_t structure

phaseref The already processed phase reference bpmproc_t structure

mode A bitpattern encoding what exactly to process

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 10 of file process_dipole.c.

References bpm_error(), correct_gain(), bpmconf::name, postprocess_waveform(), and process_-waveform().

6.10.7.4 EXTERN int process_waveform (doublewf_t * *signal*, bpmconf_t * *bpm*, bpmproc_t * *proc*, bpmproc_t * *trig*, unsigned int *mode*)

Top-level routine to processes a BPM beam pulse waveform (decaying "sin"-like wave) and derive amplitude and phase from the signal. The routine needs to be fed with a **doublewf_t** (p. 141) containing the digitized signal. The signal is checked for saturation, it's pedestal is determined and removed, the pulse starttime (t0) is set from the configuration or the trigger. Then, depending on the mode bitpattern, an FFT is performed, the waveform is fitted and a digital downconversion is done. The results (amplitude and phase) are stored in the bpmproc_t structure of the BPM.

Relevant mode bit patterns for this routine are :

- PROC_DO_FFT : The Fourier Transform of the waveform gets computed and stored as a **complexwf_t** (p. 140) in the **bpmproc_t::ft** (p. 134) variable.
- PROC_FIT_FFT : An attempt to fit the Fourier Transform is made using a Lorentizan Lineshape. If successfull, the **bpmproc_t::fft_freq** (p. 134) and **bpmproc_t::fft_tdecay** (p. 134) variables will contain the fitted frequency and decaytime. I recommend however to use a 3th party fitting routine for this (e.g. MINUIT) and implement this in a user program.
- PROC_DO_FIT : Attempts to fit a decaying sine wave to the waveform having the frequency, the decay time, the amplitude and phase as free parameters. If successfull, the **bpmproc_t::fit_freq** (p. 137), **bpmproc_t::fit_amp** (p. 136), **bpmproc_t::fit_phase** (p. 136) and **bpmproc_t::fit_tdecay** (p. 137) will contain the fit parameters. Again, I recommend to use a 3th party fitting routine for this.
- PROC_DO_DDC : Will perform a digital downconversion on the waveform. The results are contained in **bpmproc_t::ddc_amp** (p. 135) and **bpmproc_t::ddc_phase** (p. 135), determined at **bpmproc_t::ddc_tSample** (p. 135), but extrapolated back to **bpmproc_t::t0** (p. 133).
- PROC_DDC_FITTDECAY, PROC_DDC_FFTTDECAY : Normally the ddc algoritm gets it's decay time for extrapolation back to t0 from the **bpmconf_t::ddc_tdecay** (p. 128) variable, if one of these flags are set it will get them from the fitted waveform or FFT if they were succesful.
- PROC_DDC_FITFREQ, PROC_DDC_FFTFREQ : Analogous as the previous item, but now for the ddc frequency which is normally obtained from **bpmconf_t::ddc_freq** (p. 128).
- PROC_DDC_FULL : Will perform the DDC algorithm on the entire waveform and store the result in **bpmproc_t::dc** (p. 134)

Parameters:

signal The digitized signal converted into a **doublewf_t** (p. 141)

bpm A pointer to the bpmconf_t structure for the BPM channel

proc A pointer to the bpmproc_t structure for the BPM channel

trig A pointer to the bpmproc_t structure of the trigger for this BPM channel, if this parameter is NULL, externall clocking will be assumed and the t0 from the bpmconf_t structure will be used in the processing.

mode The processing mode bitword

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 12 of file process_waveform.c.

References bpmproc::ampnoise, bpm_error(), bpm_warning(), bpmconf::cav_decaytime, check_-saturation(), bpmproc::dc, bpmproc::ddc_amp, bpmconf::ddc_buffer_im, bpmconf::ddc_buffer_re, bpmconf::ddc_filter, bpmconf::ddc_freq, bpmproc::ddc_iSample, bpmproc::ddc_phase, ddc_sample_-waveform(), bpmproc::ddc_success, bpmconf::ddc_tdecay, bpmconf::ddc_tOffset, bpmproc::ddc_-tSample, ddc_waveform(), bpmconf::digi_freq, bpmconf::digi_nbits, bpmconf::digi_nsamples, doublewf_bias(), bpmproc::fft_freq, bpmproc::fft_success, bpmproc::fft_tdecay, fft_waveform(), bpmproc::fit_amp, fit_fft(), bpmproc::fit_freq, bpmproc::fit_phase, bpmproc::fit_success, bpmproc::fit_-tdecay, bpmconf::fit_tOffset, fit_waveform(), bpmproc::ft, get_pedestal(), bpmproc::iunsat, bpm-conf::name, norm_phase(), bpmproc::saturated, bpmconf::t0, bpmproc::t0, bpmproc::voltageoffset, and complexwf_t::wf.

Referenced by process_dipole(), and process_monopole().

6.10.7.5 EXTERN int postprocess_waveform (bpmconf_t * *bpm*, bpmproc_t * *proc*, bpmcalib_t * *cal*, bpmproc_t * *ampref*, bpmproc_t * *phaseref*, unsigned int *mode*)

Top-level routine to Post-process a waveform for whith the amplitude and the phase have already been defined using process_waveform. This routine goes on to calculate I and Q from the phase and amplitudes as well as the postion and slope using the calibration information.

Relevant mode bit patterns for this routine are :

- PROC_RAW_PHASE : when this bit is active in the mode word, the routine will not replace the phase in the bpmproc_t structure by the phase difference between the reference cavity and the processed cavity. Under normal circumstances you don't want this since it's only the phase difference which actually has any physical meaning.

Parameters:

signal The digitized signal converted into a **doublewf_t** (p. 141)

bpm A pointer to the bpmconf_t structure for the BPM channel

proc A pointer to the bpmproc_t structure for the BPM channel

cal A pointer to the bpmcalib_t structure for the BPM channel

ampref A pointer to the bpmproc_t structure of the amplitude reference channel for this BPM.

phaseref A pointer to the bpmproc_t structure of the phase reference channel for this BPM.

mode The processing mode bitword

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 10 of file postprocess_waveform.c.

References bpm_error(), bpmproc::ddc_amp, bpmproc::ddc_I, bpmcalib::ddc_IQphase, bpmproc::ddc_-phase, bpmproc::ddc_pos, bpmcalib::ddc_posscale, bpmproc::ddc_Q, bpmproc::ddc_slope, bpmcalib::ddc_slopescale, bpmproc::ddc_success, bpmproc::fit_amp, bpmproc::fit_I, bpmcalib::fit_-IQphase, bpmproc::fit_phase, bpmproc::fit_pos, bpmcalib::fit_posscale, bpmproc::fit_Q, bpmproc::fit_-slope, bpmcalib::fit_slopescale, bpmproc::fit_success, get_IQ(), get_pos(), get_slope(), bpmconf::name, and norm_phase().

Referenced by process_dipole().

6.10.7.6 EXTERN int process_caltone (doublewf_t * *signal*, bpmconf_t * *bpm*, bpmproc_t * *proc*, unsigned int *mode*)

Top level routine to process the calibration tone via DDC, similar to process_waveform but it also updates the ddc_ct_amp and ddc_ct_phase variables in the bpmproc_t structure. No fitting is implemented in this routine.

Relevant mode bit patterns for this routine are analogous as in process_waveform

- PROC_DO_FFT : see process_waveform
- PROC_FIT_FFT : see process_waveform
- PROC_DO_DDC : see process_waveform

Parameters:

signal The digitized signal converted into a **doublewf_t** (p. 141)

bpm A pointer to the bpmconf_t structure for the BPM channel

proc A pointer to the bpmproc_t structure for the BPM channel

mode The processing mode bitword

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 11 of file process_caltone.c.

References bpmproc::ampnoise, bpm_error(), bpm_warning(), check_saturation(), bpmproc::dc, bpmproc::ddc_amp, bpmconf::ddc_buffer_im, bpmconf::ddc_buffer_re, bpmproc::ddc_ct_amp, bpmconf::ddc_ct_filter, bpmconf::ddc_ct_freq, bpmconf::ddc_ct_iSample, bpmproc::ddc_ct_phase, bpmproc::ddc_phase, bpmproc::ddc_success, ddc_waveform(), bpmconf::digi_nbits, doublewf_bias(), bpmproc::fft_freq, bpmproc::fft_success, bpmproc::fft_tdecay, fft_waveform(), fit_fft(), bpmproc::ft, get_-pedestal(), bpmproc::iunsat, bpmconf::name, norm_phase(), bpmproc::saturated, bpmproc::voltageoffset, and complexwf_t::wf.

6.10.7.7 EXTERN int correct_gain (bpmproc_t $*$ *proc*, bpmcalib_t $*$ *cal*, unsigned int *mode*)

Correct the processed amplitude and phase by using calibration tone information if the ddc and or fits were successfull. Since e.g. for internal clock it is not really sure the phase information can be used if there is no proper trigger, some mode bits can be flagged to only correct the amplitude.

Relevant mode bit patterns for this routine are :

- PROC_CORR_AMP : Correct the amplitude
- PROC_CORR_PHASE : Correct the phase
- PROC_CORR_GAIN : Correct both of them

Parameters:

proc The bpmproc_t structure of the bpm

cal The bpmcalib_t structure of the bpm

mode Mode of correction

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 10 of file correct_gain.c.

References bpm_error(), bpmproc::ddc_amp, bpmcalib::ddc_ct_amp, bpmproc::ddc_ct_amp, bpmcalib::ddc_ct_phase, bpmproc::ddc_ct_phase, bpmproc::ddc_phase, bpmproc::ddc_success, bpmproc::fit_amp, bpmcalib::fit_ct_amp, bpmproc::fit_ct_amp, bpmcalib::fit_ct_phase, bpmproc::fit_ct_-phase, bpmproc::fit_phase, and bpmproc::fit_success.

Referenced by process_dipole(), and process_monopole().

6.10.7.8 EXTERN int fit_waveform (doublewf_t $*$ *w*, double *t0*, 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. 183) !

Attention:

Note that this routine is highly experimental, so don't use it for real production stuff. Instead I recommend using a proper minimisation package like MINUIT or so...

Parameters:

w The waveform encoded as a **doublewf_t** (p. 141)
t0 t0 for the waveform
i_freq Initial frequency for the fit
i_tdecay Initial decay time for the fit
i_amp Initial amplitude for the fit
i_phase Initial phase for the fit
freq Fitted frequency
tdecay Fitted decay time
amp Fitted amplitude
phase Fitted phase

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 80 of file fit_waveform.c.

References bpm_error(), doublewf(), doublewf_delete(), doublewf_t::fs, doublewf_t::ns, and doublewf_t::wf.

Referenced by process_waveform().

6.10.7.9 EXTERN int fit_diodepulse (doublewf_t * *w*, double * *t0*)

Fits the diode pulse, basically a wrapper for get_t0, to conserve names and consistency in the library... is nothing more than a wrapper around get_t0, so see there...

Definition at line 10 of file fit_diodepulse.c.

References get_t0().

Referenced by process_diode().

6.10.7.10 EXTERN int fft_waveform (doublewf_t * *w*, complexwf_t * *ft*)

Performs a fast fourier transform of the waveform, after subtracting the pedestal, basically just a wrapper around the forward realfft routine from the DSP module. Please see it's documentation for more details...

Parameters:

w the waveform
fft the complex returned fft spectrum

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 12 of file fft_waveform.c.

References bpm_error(), FFT_FORWARD, and realfft().

Referenced by process_caltone(), and process_waveform().

6.10.7.11 EXTERN int fit_fft_prepare (complexwf_t ∗ *ft*, int ∗ *n1*, int ∗ *n2*, double ∗ *amp*, double ∗ *freq*, double ∗ *fwhm*)

This routine prepares the fft fit of the waveform. It starts by getting the position of the maximum in the spectrum (first nyquist band only). Then from this position runs left and right to determine where the amplitude drops to half of the peak amplitude and have an initial estimation of the FWHM. It will then set twice the FWHM width as the fit range in which to perform the fit, this is than returned by the samplnumbers n1 and n2.

Parameters:

ft The **complexwf_t** (p. 140) fourier transform

n1 The first sample to start the fit from

n2 The last sample to take into account in the following fit

amp Initial estimation of the amplitude for the fit

freq Initial estimation of the frequency for the fit

fwhm Initial estimation of the FWHM for the fit.

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure.

Definition at line 72 of file fit_fft.c.

References bpm_error(), complexwf_t::fs, complexwf_t::ns, and complexwf_t::wf.

Referenced by fit_fft().

6.10.7.12 EXTERN int fit_fft (complexwf_t ∗ *ft*, double ∗ *freq*, double ∗ *tdecay*, double ∗ *A*, double ∗ *C*)

Fits the power spectrum of the FT of a waveform frequency and decay time. Internally it makes a call to fit_fft_prepare to get an initial estimation of the parameters and goes on by applying the nr_lmder routine to minimise the fourier transform power spectrum agains a lorentzian lineshape defined by

$$L = \frac{p_0}{(f - p_1)^2 + \left(\frac{p_2}{2}\right)^2} + p_3$$

Where

- p0 = the amplitude of the power spectrum
- p1 = the frequency of the fourier transform peak
- p2 = the full width at half maximum
- p3 = a constant offset

Parameters:

ft The **complexwf_t** (p. 140) encoded fourier transform

freq The returned frequency (p1)

tdecay The returned tdecay (p2)

a p0 (amplitude of powerspectrum) of the fit (can be NULL if not interested)

c p3 (offset) of the fit (can be NULL if not interested)

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 148 of file fit_fft.c.

References bpm_error(), fit_fft_prepare(), complexwf_t::fs, complexwf_t::ns, and complexwf_t::wf.

Referenced by process_caltone(), and process_waveform().

6.10.7.13 EXTERN int check_saturation (doublewf_t ∗ *w*, int *nbits*, int ∗ *iunsat*)

Checks the saturation, so computes the first sample where no saturation occurs. If no saturation occurred in the waveform, this sample - stored in iunsat - will be set to 0. A saturated sample is found when it's ADC value is more (resp. less) than then maximum allowed ADC value (2^nbits) minus a threshold set to 15. (resp. the minium allowed ADC value, being 0) plus a threshold set to 15.

Attention:

The waveform contained in the **doublewf_t** (p. 141) SHOULD NOT have been pedestal corrected. This routine will assume the waveform runs between 0 and 2^nbits.

Note the return code of the routine is slightly different than whan is conventional in libbpm since I wanted to encode whether saturation was found or not as the return code of the routine.

Parameters:

w The waveform to check, encoded as a **doublewf_t** (p. 141)

nbits The number of digitiser bits (e.g. 12 or 14)

iunsat The returned last unsaturated sample

Returns:

1 when saturation was present, 0 when not, -1 when failure occurred

Definition at line 11 of file check_saturation.c.

References bpm_error(), doublewf_t::ns, and doublewf_t::wf.

Referenced by process_caltone(), and process_waveform().

6.10.7.14 EXTERN int downmix_waveform (doublewf_t ∗ *w*, double *frequency*, complexwf_t ∗ *out*)

Downmixes the input waveform agains a complex LO using a frequency f and phase 0, the real part of the resulting complex waveform was mixed against a cosine-like wave, the imaginary part against a sinus-like. Note that this is just the downmixing itself, no filtering whatsoever is applied here.

Parameters:

w The input waveform, encoded as a **doublewf_t** (p. 141)

freq The frequency of the digital LO

out The complex output downmixed waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure.

Definition at line 10 of file downmix_waveform.c.

References bpm_error(), doublewf_t::fs, complex_t::im, doublewf_t::ns, complex_t::re, doublewf_t::wf, and complexwf_t::wf.

6.10.7.15 EXTERN int ddc_waveform (doublewf_t ∗ *w*, double *frequency*, filter_t ∗ *filt*, complexwf_t ∗ *dc*, doublewf_t ∗ *buf_re*, doublewf_t ∗ *buf_im*)

As this is a pure wrapper around the ddc routine out of the dsp packate, please see the documentation there.

Definition at line 12 of file ddc_waveform.c.

References bpm_error(), and ddc().

Referenced by process_caltone(), and process_waveform().

6.10.7.16 EXTERN int ddc_sample_waveform (doublewf_t ∗ *w*, double *frequency*, filter_t ∗ *filt*, int *iSample*, double *t0*, double *tdecay*, double ∗ *amp*, double ∗ *phase*, doublewf_t ∗ *buf_re*, doublewf_t ∗ *buf_im*)

TO BE IMPLEMENTED !!!

This routine will contain a quicker version of the ddc algorithm that doesn't filter the entire waveform and only applies the filter at the sampling point. However, I need to make custom a apply_filter routine which is universally valid for all types of filters (IIR as well).

Definition at line 19 of file ddc_sample_waveform.c.

References bpm_error().

Referenced by process_waveform().

6.10.7.17 EXTERN int get_pedestal (doublewf_t ∗ *wf*, int *range*, double ∗ *offset*, double ∗ *rms*)

Find the mean pedestal using the first 20 (or how ever many are required) sample values, store the results in the offset and rms. This routine in fact just calls the doublewf_basic_stats routine and gets the appropriate values from the **wfstat_t** (p. 153) structure.

Parameters:

- ***wf*** The signal encoded as a **doublewf_t** (p. 141)
- ***range*** The maximum sample to go to average over. The pedestal gets determined from the first "range" samples of the waveform
- ∗***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 10 of file get_pedestal.c.

References bpm_error(), doublewf_basic_stats(), wfstat_t::mean, and wfstat_t::rms.

Referenced by get_t0(), process_caltone(), and process_waveform().

6.10.7.18 EXTERN int get_t0 (doublewf_t * *w*, double * *t0*)

Finds the t0 value from a diode peak, used in the case of internall triggering when a trigger pulse needs to be specified to calculate beam arrival

Attention:

This routine needs some optimisation in terms of speed and some general checking in terms of correctness. Probably some re-writing using the bpmwf structures would be good...

Parameters:

w A pointer to the **doublewf_t** (p. 141) signal

t0 returns t0

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 46 of file get_t0.c.

References bpm_error(), bpm_verbose, bpm_warning(), doublewf_t::fs, get_pedestal(), nr_fit(), doublewf_t::ns, and doublewf_t::wf.

Referenced by fit_diodepulse().

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

Gets the I and Q from the amplitude and phase of the waveform and it's respective references. The I and Q are calculated respectively as :

$$I = \frac{A}{A_{ref}} \cos(\phi - \phi_{ref})$$

and

$$Q = \frac{A}{A_{ref}} \sin(\phi - \phi_{ref})$$

Parameters:

amp The amplitude of the considered waveform

phase The phase of the considered waveform

refamp The amplitude of the reference cavity

refphase The phase of the reference cavity

Q The returned Q value

I The returned I value

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file get_IQ.c.

References bpm_error(), and bpm_warning().

Referenced by postprocess_waveform().

6.10.7.20 EXTERN int get_pos (double *Q*, double *I*, double *IQphase*, double *posscale*, double $*$ *pos*)

Returns the beam given I and Q values, IQphase and scale, it is calcualted as

$$x = c\,[I\cos(\phi_{IQ}) + Q\sin(\phi_I Q)]$$

Where c is the positionscale and x the position.

Parameters:

- ***Q*** The Q value (obtained from get_IQ)
- ***I*** The I value (obtained from get_IQ)
- ***IQphase*** The IQ phase rotation
- ***posscale*** The position scale
- ***pos*** The returned position

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file get_pos.c.

References bpm_error().

Referenced by postprocess_waveform().

6.10.7.21 EXTERN int get_slope (double *Q*, double *I*, double *IQphase*, double *slopescale*, double $*$ *slope*)

Returns the beam slope given I and Q values, IQphase and scale, it is calcualted as

$$x' = c\,[-I\sin(\phi_{IQ}) + Q\cos(\phi_I Q)]$$

Where c is the positionscale and x the position.

Parameters:

- ***Q*** The Q value (obtained from get_IQ)
- ***I*** The I value (obtained from get_IQ)
- ***IQphase*** The IQ phase rotation
- ***slopescale*** The slope scale
- ***slope*** The returned slope

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file get_slope.c.

References bpm_error().

Referenced by postprocess_waveform().

6.11 Waveform handling routines

6.11.1 Detailed Description

This module contains the basic waveform handling routines and structures for libbpm

The bpmwf sublibrary implements 3 waveform types **doublewf_t** (p. 141), **intwf_t** (p. 149) and **complexwf_t** (p. 140), all of which are simple structure typedefs which hold the number of samples, the sampling frequency and a pointer "wf" to the waveform. So the data array is accessible via **doublewf_t::wf** (p. 142) as a normal array of integers, doubles and **complex_t** (p. 140) 's.

6.11.2 Memory management

All have memory management routines (allocation/deletion) and routines to cast to other times (eg **doublewf_t** (p. 141) -> **intwf_t** (p. 149) or the other way around). This can be done either by filling existing waveforms (convenient when you e.g. have already allocated memory and referenced it into a root branch) or by having the casting routine allocate memory itself and return a pointer to it. e.g:

```
intwf_t *w = intwf_cast_new( doublewf_t *dw );
```

this allocates memory for **intwf_t** (p. 149) and returns a pointer it, or

```
intwf_cast( intwf_t *w, doublewf_t *dw );
```

this casts dw into existing intwf w.

The sublibrary employs the sampling convention, where the sample is taken at the time index corresponding to

```
t = (double) i / sampling_freq
```

6.11.3 Waveform handling

The sublibrary implements basic waveform hanlding like addition, subtraction, multiplication, division, biasing and scaling.

Some advanced routines like differentiation, integration of the waveforms are also present. Also interpolation is impemented using various schemes which are more applicable depending on the type of waveform : linear, parabolic : for non repeatative signals, sinc and lanczos for repeatative signals (cfr. Shannon-Whittaker interpolation). (thinking of cubic-spline as well... but not implemented yet). Using these interpolation schemes, the sublibrary also implements resampling routines.

The complex waveforms have a set of routines to extract real/imag parts as well as phase and amplitude. Similar comments apply as for the casting routines, where the "_new" versions allocate memory in the routine and return a pointer to it.

6.11.4 Filling the waveforms

The values of the waveforms can be set by either filling them from a given array of values using e.g.

```
doublewf_setvalues( doublewf_t *w, double *a)
```

or by calculating them from a function which returns the basic type of the waveform.

E.g. define a complex valued function in your code:

```
complex_t csin( double t, int npars, double a ) {
  complex_t z
  // calculate a complex number z from the time t and parameters...
  return z;
}
```

which returns a complex value from the time `t` and having `npar` paramaters `a`[0] ... `a`[n-1]

You can fill a waveform (and so bascially sample the function at sampling frequency fs) by executing

```
complexwf_setfunction( complexwf_t *z, &csin, npars, a )
```

Also some routines are added to fill the waveforms with CW tones and decaying waves, along with some noise adding routines etc...

6.11.5 Note on the interpolation options.

Here are some examples of the different interpolation options that one can give to the doublewf/complexwf_getvalue() or _resample() routines.

6.11.6 For examples...

For examples on library use, please see the examples/wf directory in the libbpm main tree...

6.11.7 Todo list

- implement cubic spline interpolation ?

Files

- file **bpm_wf.h**

  Simple waveform handling routines for libbpm.

- file **complexwf.c**
- file **doublewf.c**
- file **intwf.c**
- file **wfstats.c**

Data Structures

- struct **doublewf_t**
- struct **intwf_t**
- struct **complexwf_t**
- struct **wfstat_t**

Defines

- #define **WF_EPS**
- #define **MAX_ALLOWED_NS**
- #define **WF_NEAREST**
- #define **WF_LINEAR**
- #define **WF_QUADRATIC**
- #define **WF_SINC**
- #define **WF_LANCZOS**

Functions

- EXTERN int **wfstat_reset** (**wfstat_t** ∗s)
- EXTERN void **wfstat_print** (FILE ∗of, **wfstat_t** ∗s)
- EXTERN **doublewf_t** ∗ **doublewf** (int ns, double fs)
- EXTERN **doublewf_t** ∗ **doublewf_time_series** (int ns, double fs)
- EXTERN **doublewf_t** ∗ **doublewf_sample_series** (int ns, double fs)
- EXTERN **doublewf_t** ∗ **doublewf_frequency_series** (int ns, double fs)
- EXTERN int **doublewf_setvalues** (**doublewf_t** ∗w, double ∗x)
- EXTERN int **doublewf_setfunction** (**doublewf_t** ∗w, double(∗wffun)(double t, int, double ∗), int npars, double ∗par)
- EXTERN int **doublewf_copy** (**doublewf_t** ∗copy, **doublewf_t** ∗src)
- EXTERN **doublewf_t** ∗ **doublewf_copy_new** (**doublewf_t** ∗w)
- EXTERN int **doublewf_subset** (**doublewf_t** ∗sub, **doublewf_t** ∗w, int i1, int i2)
- EXTERN int **doublewf_reset** (**doublewf_t** ∗w)
- EXTERN void **doublewf_delete** (**doublewf_t** ∗w)
- EXTERN **intwf_t** ∗ **intwf_cast_new** (**doublewf_t** ∗w)
- EXTERN int **intwf_cast** (**intwf_t** ∗iw, **doublewf_t** ∗w)
- EXTERN int **doublewf_compat** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_add** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_subtract** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_multiply** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_divide** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_scale** (double f, **doublewf_t** ∗w)
- EXTERN int **doublewf_bias** (double c, **doublewf_t** ∗w)
- EXTERN int **doublewf_add_cwtone** (**doublewf_t** ∗w, double amp, double phase, double freq, double phasenoise)
- EXTERN int **doublewf_add_dcywave** (**doublewf_t** ∗w, double amp, double phase, double freq, double ttrig, double tdcy, double phasenoise)
- EXTERN int **doublewf_add_ampnoise** (**doublewf_t** ∗w, double sigma)
- EXTERN int **doublewf_basic_stats** (**doublewf_t** ∗w, int s0, int s1, **wfstat_t** ∗stats)
- EXTERN int **doublewf_derive** (**doublewf_t** ∗w)
- EXTERN int **doublewf_integrate** (**doublewf_t** ∗w)
- EXTERN void **doublewf_print** (FILE ∗of, **doublewf_t** ∗w)
- EXTERN double **doublewf_getvalue** (**doublewf_t** ∗w, double t, unsigned int mode)
- EXTERN int **doublewf_resample** (**doublewf_t** ∗w2, double fs, **doublewf_t** ∗w1, unsigned int mode)
- EXTERN **intwf_t** ∗ **intwf** (int ns, double fs)
- EXTERN **intwf_t** ∗ **intwf_sample_series** (int ns, double fs)
- EXTERN int **intwf_setvalues** (**intwf_t** ∗w, int ∗x)

- EXTERN int **intwf_setfunction** (**intwf_t** ∗w, int(∗wffun)(double t, int, double ∗), int npars, double ∗par)
- EXTERN int **intwf_copy** (**intwf_t** ∗copy, **intwf_t** ∗src)
- EXTERN **intwf_t** ∗ **intwf_copy_new** (**intwf_t** ∗w)
- EXTERN int **intwf_subset** (**intwf_t** ∗sub, **intwf_t** ∗w, int i1, int i2)
- EXTERN int **intwf_reset** (**intwf_t** ∗w)
- EXTERN void **intwf_delete** (**intwf_t** ∗w)
- EXTERN **doublewf_t** ∗ **doublewf_cast_new** (**intwf_t** ∗w)
- EXTERN int **doublewf_cast** (**doublewf_t** ∗w, **intwf_t** ∗iw)
- EXTERN int **intwf_compat** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_add** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_subtract** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_multiply** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_divide** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_scale** (int f, **intwf_t** ∗w)
- EXTERN int **intwf_bias** (int c, **intwf_t** ∗w)
- EXTERN int **intwf_add_cwtone** (**intwf_t** ∗w, double amp, double phase, double freq, double phasenoise)
- EXTERN int **intwf_add_dcywave** (**intwf_t** ∗w, double amp, double phase, double freq, double ttrig, double tdcy, double phasenoise)
- EXTERN int **intwf_add_ampnoise** (**intwf_t** ∗w, double sigma)
- EXTERN int **intwf_basic_stats** (**intwf_t** ∗w, int s0, int s1, **wfstat_t** ∗stats)
- EXTERN int **intwf_derive** (**intwf_t** ∗w)
- EXTERN int **intwf_integrate** (**intwf_t** ∗w)
- EXTERN void **intwf_print** (FILE ∗of, **intwf_t** ∗w)
- EXTERN int **intwf_getvalue** (**intwf_t** ∗w, double t, unsigned int mode)
- EXTERN int **intwf_resample** (**intwf_t** ∗w2, double fs, **intwf_t** ∗w1, unsigned int mode)
- EXTERN **complexwf_t** ∗ **complexwf** (int ns, double fs)
- EXTERN **complexwf_t** ∗ **complexwf_copy_new** (**complexwf_t** ∗w)
- EXTERN int **complexwf_copy** (**complexwf_t** ∗copy, **complexwf_t** ∗src)
- EXTERN int **complexwf_subset** (**complexwf_t** ∗sub, **complexwf_t** ∗w, int i1, int i2)
- EXTERN int **complexwf_setvalues** (**complexwf_t** ∗w, **complex_t** ∗x)
- EXTERN int **complexwf_setfunction** (**complexwf_t** ∗w, **complex_t**(∗wffun)(double, int, double ∗), int npars, double ∗par)
- EXTERN int **complexwf_reset** (**complexwf_t** ∗w)
- EXTERN void **complexwf_delete** (**complexwf_t** ∗w)
- EXTERN int **complexwf_compat** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_add** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_subtract** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_multiply** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_divide** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_scale** (**complex_t** f, **complexwf_t** ∗w)
- EXTERN int **complexwf_bias** (**complex_t** c, **complexwf_t** ∗w)
- EXTERN int **complexwf_add_cwtone** (**complexwf_t** ∗w, double amp, double phase, double freq, double phasenoise)
- EXTERN int **complexwf_add_dcywave** (**complexwf_t** ∗w, double amp, double phase, double freq, double ttrig, double tdcy, double phasenoise)
- EXTERN int **complexwf_add_noise** (**complexwf_t** ∗w, double sigma)
- EXTERN int **complexwf_add_ampnoise** (**complexwf_t** ∗w, double sigma)
- EXTERN int **complexwf_add_phasenoise** (**complexwf_t** ∗w, double sigma)

- EXTERN void **complexwf_print** (FILE ∗of, **complexwf_t** ∗w)
- EXTERN int **complexwf_getreal** (**doublewf_t** ∗re, **complexwf_t** ∗z)
- EXTERN int **complexwf_getimag** (**doublewf_t** ∗im, **complexwf_t** ∗z)
- EXTERN int **complexwf_getamp** (**doublewf_t** ∗r, **complexwf_t** ∗z)
- EXTERN int **complexwf_getphase** (**doublewf_t** ∗theta, **complexwf_t** ∗z)
- EXTERN **doublewf_t** ∗ **complexwf_getreal_new** (**complexwf_t** ∗z)
- EXTERN **doublewf_t** ∗ **complexwf_getimag_new** (**complexwf_t** ∗z)
- EXTERN **doublewf_t** ∗ **complexwf_getamp_new** (**complexwf_t** ∗z)
- EXTERN **doublewf_t** ∗ **complexwf_getphase_new** (**complexwf_t** ∗z)
- EXTERN int **complexwf_setreal** (**complexwf_t** ∗z, **doublewf_t** ∗re)
- EXTERN int **complexwf_setimag** (**complexwf_t** ∗z, **doublewf_t** ∗im)
- EXTERN int **time_to_sample** (double fs, int ns, double t, int ∗iS)
- EXTERN int **freq_to_sample** (double fs, int ns, double f, int ∗iS)
- EXTERN int **sample_to_time** (double fs, int ns, int iS, double ∗t)
- EXTERN int **sample_to_freq** (double fs, int ns, int iS, double ∗f)

6.11.8 Define Documentation

6.11.8.1 #define WF_EPS

A small number

Definition at line 157 of file bpm_wf.h.

Referenced by complexwf_compat(), doublewf_compat(), and intwf_compat().

6.11.8.2 #define MAX_ALLOWED_NS

Maximum allowed number of samples (2^18)

Definition at line 158 of file bpm_wf.h.

Referenced by complexwf(), doublewf(), doublewf_resample(), intwf(), and intwf_resample().

6.11.8.3 #define WF_NEAREST

No interpolation, return nearest sample

Definition at line 160 of file bpm_wf.h.

6.11.8.4 #define WF_LINEAR

Perform linear interpolation in XXXwf_getsample()

Definition at line 161 of file bpm_wf.h.

Referenced by doublewf_getvalue().

6.11.8.5 #define WF_QUADRATIC

Perform quadratic (parabolic) interpolation

Definition at line 162 of file bpm_wf.h.

Referenced by doublewf_getvalue(), and generate_bpmsignal().

6.11.8.6 #define WF_SINC

signal reconstruction using sinc kernel (0..ns)

Definition at line 163 of file bpm_wf.h.

Referenced by doublewf_getvalue().

6.11.8.7 #define WF_LANCZOS

signal reconstruction using lanczos kernel (a=3)

Definition at line 164 of file bpm_wf.h.

Referenced by doublewf_getvalue().

6.11.9 Function Documentation

6.11.9.1 EXTERN int wfstat_reset (wfstat_t * *s*)

Reset the waveform statistics structure.

Parameters:

- ***s*** A pointer to a **wfstat_t** (p. 153) structure

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 8 of file wfstats.c.

References bpm_error(), wfstat_t::imax, wfstat_t::imin, wfstat_t::max, wfstat_t::mean, wfstat_t::min, and wfstat_t::rms.

Referenced by doublewf_basic_stats().

6.11.9.2 EXTERN void wfstat_print (FILE * *of*, wfstat_t * *s*)

Prints the waveform statistics to the screen,

Parameters:

- ***of*** A filepointer
- ***s*** A pointer to the waveform statistics structure

Returns:

void

Definition at line 29 of file wfstats.c.

References bpm_error(), wfstat_t::imax, wfstat_t::imin, wfstat_t::max, wfstat_t::mean, wfstat_t::min, and wfstat_t::rms.

6.11.9.3 EXTERN doublewf_t* doublewf (int *ns*, double *fs*)

Allocates memory for a new waveform of doubles

Parameters:

ns The number of samples in the waveform

fs The sampling frequency of the waveform

Returns:

A pointer to the allocated waveform structure

Definition at line 8 of file doublewf.c.

References bpm_error(), doublewf_t::fs, MAX_ALLOWED_NS, doublewf_t::ns, and doublewf_t::wf.

Referenced by complexwf_getamp_new(), complexwf_getimag_new(), complexwf_getphase_new(), complexwf_getreal_new(), ddc_initialise(), doublewf_cast_new(), doublewf_copy_new(), doublewf_-frequency_series(), doublewf_sample_series(), doublewf_time_series(), fit_waveform(), generate_-bpmsignal(), generate_diodesignal(), and get_mode_response().

6.11.9.4 EXTERN doublewf_t∗ doublewf_time_series (int *ns*, double *fs*)

Allocates memory for a new waveform of doubles and fills it with the sample time values

Parameters:

ns The number of samples in the waveform

fs The sampling frequency of the waveform

Returns:

A pointer to the allocated waveform structure

Definition at line 63 of file doublewf.c.

References doublewf(), doublewf_t::fs, doublewf_t::ns, and doublewf_t::wf.

6.11.9.5 EXTERN doublewf_t∗ doublewf_sample_series (int *ns*, double *fs*)

Allocates memory for a new waveform of doubles and fills it with sample numbers.

Parameters:

ns The number of samples in the waveform

fs The sampling frequency of the waveform

Returns:

A pointer to the allocated waveform structure

Definition at line 50 of file doublewf.c.

References doublewf(), doublewf_t::ns, and doublewf_t::wf.

6.11.9.6 EXTERN doublewf_t∗ doublewf_frequency_series (int *ns*, double *fs*)

Allocates memory for a new waveform of doubles and fills it with the frequency values

Parameters:

ns The number of samples in the waveform

fs The sampling frequency of the waveform

Returns:

A pointer to the allocated waveform structure

Definition at line 76 of file doublewf.c.

References doublewf(), doublewf_t::fs, doublewf_t::ns, and doublewf_t::wf.

6.11.9.7 EXTERN int doublewf_setvalues (doublewf_t ∗ *w*, double ∗ *x*)

Fills the waveform of doubles with the values from the array x. No check is performed whether x contains enough samples, the user needs to be sure this is the case !

Parameters:

w A pointer to the waveform of doubles

x A pointer to the x values

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 151 of file doublewf.c.

References bpm_error(), doublewf_t::ns, and doublewf_t::wf.

6.11.9.8 EXTERN int doublewf_setfunction (doublewf_t ∗ *w*, double(∗)(double t, int, double ∗) *wffun*, int *npars*, double ∗ *par*)

Fills the waveform with values from the function wffun(), this function has to return a double from argument t (time) and has npars parameters given by the array ∗par. The function will be evaluated at the time t of each sample...

Parameters:

w A pointer to the waveform of doubles

wffun A pointer to the function to fill the waveform with

t The time parameter in the function

npars Number of parameters for the function

par Array of parameters for the function

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

6.11.9.9 EXTERN int doublewf_copy (doublewf_t ∗ *copy*, doublewf_t ∗ *src*)

Copies the values from existing waveform src into copy checks first whether the waveforms are compatible... This routine doesn't allocate memory internally and the waveforms should already have been created by the user...

Parameters:

copy A pointer to the copy waveform

src A pointer to the original waveform

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 106 of file doublewf.c.

References bpm_error(), doublewf_compat(), doublewf_t::ns, and doublewf_t::wf.

Referenced by rf_mixer().

6.11.9.10 EXTERN doublewf_t* doublewf_copy_new (doublewf_t * *w*)

Allocates memory and produces a copy of the waveform w;

Parameters:

w A pointer to the original waveform

Returns:

A pointer to the copy of w

Definition at line 89 of file doublewf.c.

References bpm_error(), doublewf(), doublewf_t::fs, doublewf_t::ns, and doublewf_t::wf.

6.11.9.11 EXTERN int doublewf_subset (doublewf_t * *sub*, doublewf_t * *w*, int *i1*, int *i2*)

Copies a subset from sample i1 to sample i2 (inclusive) to the sub waveform from waveform w. The routine expects the sub waveform to already exist with enough samples. (this is not checked !) The sub->fs and sub->ns will be overwritten.

Parameters:

sub Pointer to the waveform which will hold the subset

w Pointer to the original waveform

i1 First sample of w to copy

i2 Last sample of w to copy

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 127 of file doublewf.c.

References bpm_error(), doublewf_t::fs, doublewf_t::ns, and doublewf_t::wf.

6.11.9.12 EXTERN int doublewf_reset (doublewf_t * *w*)

Resets the waveform of doubles to 0.

Parameters:

w A pointer to the waveform of doubles

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 185 of file doublewf.c.

References bpm_error(), doublewf_t::ns, and doublewf_t::wf.

Referenced by generate_bpmsignal().

6.11.9.13 EXTERN void doublewf_delete (doublewf_t * *w*)

Frees up the memory used by the waveform

Parameters:

w A pointer to the waveform of doubles

Returns:

void

Definition at line 202 of file doublewf.c.

References bpm_warning(), and doublewf_t::wf.

Referenced by ddc_cleanup(), fit_waveform(), get_mode_response(), intwf_basic_stats(), intwf_-getvalue(), and intwf_resample().

6.11.9.14 EXTERN intwf_t* intwf_cast_new (doublewf_t * *w*)

Cast the waveform of doubles to a new waveform of integers. Memory is allocated inside this routine so the user just needs to have a inwf_t pointer ready.

Parameters:

w A pointer to the waveform of doubles

Returns:

A newly created **intwf_t** (p. 149) representation of the waveform of doubles

Definition at line 219 of file doublewf.c.

References bpm_error(), dround(), doublewf_t::fs, intwf(), intwf_t::ns, doublewf_t::ns, doublewf_t::wf, and intwf_t::wf.

6.11.9.15 EXTERN int intwf_cast (intwf_t * *iw*, doublewf_t * *w*)

Cast the waveform of doubles to an already existing waveform of integers.

Parameters:

iw A pointer to an existing waveform of integers

w A pointer to the waveform of doubles

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 245 of file doublewf.c.

References bpm_error(), dround(), intwf_t::ns, doublewf_t::wf, and intwf_t::wf.

6.11.9.16 EXTERN int doublewf_compat (doublewf_t * *w1*, doublewf_t * *w2*)

Checks compatiblity of the two waveforms, returns true if the number of samples and the sampling frequencies match. For the sampling frequency, it is simply checked whether they match to WF_EPS.

Parameters:

w1 A pointer to the first waveform of doubles

w2 A pointer to the second waveform of doubles

Returns:

1 if the waveforms match, 0 if not.

Definition at line 263 of file doublewf.c.

References bpm_error(), doublewf_t::fs, doublewf_t::ns, and WF_EPS.

Referenced by doublewf_add(), doublewf_copy(), doublewf_divide(), doublewf_multiply(), and doublewf_subtract().

6.11.9.17 EXTERN int doublewf_add (doublewf_t * *w1*, doublewf_t * *w2*)

Adds two waveforms of doubles w1+w2 sample per sample. The result is stored in w1.

Parameters:

w1 A pointer to the first waveform of doubles

w2 A pointer to the second waveform of doubles

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 276 of file doublewf.c.

References bpm_error(), bpm_warning(), doublewf_compat(), doublewf_t::ns, and doublewf_t::wf.

6.11.9.18 EXTERN int doublewf_subtract (doublewf_t * *w1*, doublewf_t * *w2*)

Subtracts two waveforms of doubles w1-w2 sample per sample. The result is stored in w1.

Parameters:

w1 A pointer to the first waveform of doubles

w2 A pointer to the second waveform of doubles

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 297 of file doublewf.c.

References bpm_error(), bpm_warning(), doublewf_compat(), doublewf_t::ns, and doublewf_t::wf.

6.11.9.19 EXTERN int doublewf_multiply (doublewf_t * *w1*, doublewf_t * *w2*)

Multiplies two waveforms of doubles w1*w2 sample per sample. The result is stored in w1.

Parameters:

- ***w1*** A pointer to the first waveform of doubles
- ***w2*** A pointer to the second waveform of doubles

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 317 of file doublewf.c.

References bpm_error(), bpm_warning(), doublewf_compat(), doublewf_t::ns, and doublewf_t::wf.

Referenced by rf_mixer().

6.11.9.20 EXTERN int doublewf_divide (doublewf_t * *w1*, doublewf_t * *w2*)

Divides two waveforms of doubles w1/w2 sample per sample. The result is stored in w1. When w2[i] is 0, w1[i] will be set to 0. and a warning message is printed.

Parameters:

- ***w1*** A pointer to the first waveform of doubles
- ***w2*** A pointer to the second waveform of doubles

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 338 of file doublewf.c.

References bpm_error(), bpm_warning(), doublewf_compat(), doublewf_t::ns, and doublewf_t::wf.

6.11.9.21 EXTERN int doublewf_scale (double *f*, doublewf_t * *w*)

Scales the waveform of doubles w by factor f. The result is stored in w.

Parameters:

- ***f*** The scalefactor
- ***w*** A pointer to the waveform of doubles

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 368 of file doublewf.c.

References bpm_error(), doublewf_t::ns, and doublewf_t::wf.

Referenced by get_mode_response(), and rf_amplify().

6.11.9.22 EXTERN int doublewf_bias (double *c*, doublewf_t $*$ *w*)

Biases the waveform of doubles w by a constant c. The result is stored in w.

Parameters:

c The constant bias.

w A pointer to the waveform of doubles

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 385 of file doublewf.c.

References bpm_error(), doublewf_t::ns, and doublewf_t::wf.

Referenced by process_caltone(), and process_waveform().

6.11.9.23 EXTERN int doublewf_add_cwtone (doublewf_t $*$ *w*, double *amp*, double *phase*, double *freq*, double *phasenoise*)

Adds a cosine-like CW tone to the entire waveform. The sampling time is taken on the array index, so t=(double)i/w->fs.

Parameters:

w A pointer to the waveform structure

amp Amplitude of the CW tone

phase Phase of the CW tone

freq Frequency of the CW tone

phasenoise Sigma of the gaussian phasenoise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 402 of file doublewf.c.

References bpm_error(), doublewf_t::fs, nr_rangauss(), doublewf_t::ns, and doublewf_t::wf.

Referenced by rf_addLO().

6.11.9.24 EXTERN int doublewf_add_dcywave (doublewf_t $*$ *w*, double *amp*, double *phase*, double *freq*, double *ttrig*, double *tdcy*, double *phasenoise*)

Adds a decaying wave pulse to the waveform. The sampling time is taken on the array index, so t=(double)i/w->fs. The added signal is of the form :

$$ampe^{-(t-ttrig)/tdcy}cos(2\pi freq(t-ttrig)+phase)$$

If desired, phasenoise is added to the phase of the waveform.

Parameters:

w A pointer to the waveform structure
amp Amplitude of the CW tone
phase Phase of the CW tone
freq Frequency of the CW tone
ttrig Trigger time of the pulse
tdcy Decay time of the pulse
phasenoise Sigma of the gaussian phasenoise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 422 of file doublewf.c.

References bpm_error(), doublewf_t::fs, nr_rangauss(), doublewf_t::ns, and doublewf_t::wf.

6.11.9.25 EXTERN int doublewf_add_ampnoise (doublewf_t * *w*, double *sigma*)

Adds gaussian amplitude noise to the waveform.

Parameters:

w A pointer to the waveform structure
sigma The gaussian sigma of the amplitude noise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 447 of file doublewf.c.

References bpm_error(), nr_rangauss(), doublewf_t::ns, and doublewf_t::wf.

6.11.9.26 EXTERN int doublewf_basic_stats (doublewf_t * *w*, int *s0*, int *s1*, wfstat_t * *stats*)

Retrieves some basic statistics about the waveform of doubles in w, only considers samples between s0 and s1.

Parameters:

w A pointer to the waveform structure
s0 First sample to consider
s1 Last sample to consider
stats A filled **wfstat_t** (p. 153) structure is returned.

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 467 of file doublewf.c.

References bpm_error(), bpm_warning(), wfstat_t::imax, wfstat_t::imin, wfstat_t::max, wfstat_t::mean, wfstat_t::min, doublewf_t::ns, wfstat_t::rms, doublewf_t::wf, and wfstat_reset().

Referenced by get_pedestal(), intwf_basic_stats(), and process_diode().

6.11.9.27 EXTERN int doublewf_derive (doublewf_t ∗ *w*)

Produce the derivative waveform for w : dw/dt.

Parameters:

w A pointer to the waveform structure.

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 507 of file doublewf.c.

References bpm_error(), doublewf_t::fs, doublewf_t::ns, and doublewf_t::wf.

6.11.9.28 EXTERN int doublewf_integrate (doublewf_t ∗ *w*)

Produce the integrated waveform for w : ^t w(s)ds.

Parameters:

w A pointer to the waveform structure.

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 532 of file doublewf.c.

References bpm_error(), doublewf_t::fs, doublewf_t::ns, and doublewf_t::wf.

Referenced by get_mode_response().

6.11.9.29 EXTERN void doublewf_print (FILE ∗ *of*, doublewf_t ∗ *w*)

Print the waveform to the filepointer

Parameters:

of A filepointer, use stdout for the terminal

w A pointer to the waveform

Returns:

void

Definition at line 556 of file doublewf.c.

References bpm_error(), doublewf_t::fs, doublewf_t::ns, and doublewf_t::wf.

6.11.9.30 EXTERN double doublewf_getvalue (doublewf_t ∗ *w*, double *t*, unsigned int *mode*)

Return the value for the waveform at sample time t, according to the interpolation mode.

Parameters:

w A pointer to the waveform structure

t A time at which to sample the waveform

mode Interpolation mode

Returns:

the value of the waveform at time t

Definition at line 575 of file doublewf.c.

References bpm_error(), doublewf_t::fs, lanczos(), nr_quadinterpol(), doublewf_t::ns, sinc(), doublewf_t::wf, WF_LANCZOS, WF_LINEAR, WF_QUADRATIC, and WF_SINC.

Referenced by digitise(), doublewf_resample(), generate_bpmsignal(), intwf_getvalue(), and intwf_resample().

6.11.9.31 EXTERN int doublewf_resample (doublewf_t * *w2*, double *fs*, doublewf_t * *w1*, unsigned int *mode*)

Resamples the waveform w1 into w2 with new fs sampling frequency This routine recalculates the correct number of samples required. However the user needs to make sure that there are enough samples in w2 available as this is not checked. The w2->ns value will be overwritten with the correct amount. The routine checkes whether the maximum allowed number of samples is not exceeded to avoid memory problems.

Parameters:

w A pointer to the waveform structure

t A time at which to sample the waveform

mode Interpolation mode

Returns:

the value of the waveform at time t

Definition at line 664 of file doublewf.c.

References bpm_error(), doublewf_getvalue(), doublewf_t::fs, MAX_ALLOWED_NS, doublewf_t::ns, and doublewf_t::wf.

6.11.9.32 EXTERN intwf_t* intwf (int *ns*, double *fs*)

Allocates memory for a new waveform of integers

Parameters:

ns The number of samples in the waveform

fs The sampling frequency of the waveform

Returns:

A pointer to the allocated waveform structure

Definition at line 8 of file intwf.c.

References bpm_error(), intwf_t::fs, MAX_ALLOWED_NS, intwf_t::ns, and intwf_t::wf.

Referenced by intwf_cast_new(), intwf_copy_new(), and intwf_sample_series().

6.11.9.33 EXTERN intwf_t* intwf_sample_series (int *ns*, double *fs*)

Allocates memory for a new waveform of integers and fills it with sample numbers.

Parameters:

ns The number of samples in the waveform

fs The sampling frequency of the waveform

Returns:

A pointer to the allocated waveform structure

Definition at line 50 of file intwf.c.

References intwf(), intwf_t::ns, and intwf_t::wf.

6.11.9.34 EXTERN int intwf_setvalues (intwf_t * *w*, int * *x*)

Fills the waveform of integers with the values from the array x. No check is performed whether x contains enough samples, the user needs to be sure this is the case !

Parameters:

w A pointer to the waveform of integers

x A pointer to the x values

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 126 of file intwf.c.

References bpm_error(), intwf_t::ns, and intwf_t::wf.

6.11.9.35 EXTERN int intwf_setfunction (intwf_t * *w*, int(*)(double t, int, double *) *wffun*, int *npars*, double * *par*)

Fills the waveform with values from the function wffun(), this function has to return a double from argument t (time) and has npars parameters given by the array *par. The function will be evaluated at the time t of each sample...

Parameters:

w A pointer to the waveform of integers

wffun A pointer to the function to fill the waveform with

t The time parameter in the function

npars Number of parameters for the function

par Array of parameters for the function

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

6.11.9.36 EXTERN int intwf_copy (intwf_t * *copy*, intwf_t * *src*)

Copies the values from existing waveform src into copy checks first whether the waveforms are compatible... This routine doesn't allocate memory internally and the waveforms should already have been created by the user...

Parameters:

copy A pointer to the copy waveform

src A pointer to the original waveform

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 81 of file intwf.c.

References bpm_error(), intwf_compat(), intwf_t::ns, and intwf_t::wf.

6.11.9.37 EXTERN intwf_t* intwf_copy_new (intwf_t * *w*)

Allocates memory and produces a copy of the waveform w;

Parameters:

w A pointer to the original waveform

Returns:

A pointer to the copy of w

Definition at line 63 of file intwf.c.

References bpm_error(), intwf_t::fs, intwf(), intwf_t::ns, and intwf_t::wf.

6.11.9.38 EXTERN int intwf_subset (intwf_t * *sub*, intwf_t * *w*, int *i1*, int *i2*)

Copies a subset from sample i1 to sample i2 (inclusive) to the sub waveform from waveform w. The routine expects the sub waveform to already exist with enough samples. (this is not checked !) The sub->fs and sub->ns will be overwritten.

Parameters:

sub Pointer to the waveform which will hold the subset

w Pointer to the original waveform

i1 First sample of w to copy

i2 Last sample of w to copy

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 102 of file intwf.c.

References bpm_error(), intwf_t::fs, intwf_t::ns, and intwf_t::wf.

6.11.9.39 EXTERN int intwf_reset (intwf_t * *w*)

Resets the waveform of integers to 0.

Parameters:

w A pointer to the waveform of integers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 160 of file intwf.c.

References bpm_error(), intwf_t::ns, and intwf_t::wf.

6.11.9.40 EXTERN void intwf_delete (intwf_t * *w*)

Frees up the memory used by the waveform

Parameters:

w A pointer to the waveform of integers

Returns:

void

Definition at line 177 of file intwf.c.

References bpm_warning(), and intwf_t::wf.

6.11.9.41 EXTERN doublewf_t* doublewf_cast_new (intwf_t * *w*)

Cast the waveform of integers to a new waveform of doubles. Memory is allocated inside this routine so the user just needs to have a inwf_t pointer ready.

Parameters:

w A pointer to the waveform of integers

Returns:

A newly created **doublewf_t** (p. 141) representation of the waveform of integers

Definition at line 194 of file intwf.c.

References bpm_error(), doublewf(), intwf_t::fs, intwf_t::ns, intwf_t::wf, and doublewf_t::wf.

Referenced by intwf_basic_stats(), intwf_getvalue(), and intwf_resample().

6.11.9.42 EXTERN int doublewf_cast (doublewf_t * *w*, intwf_t * *iw*)

Cast the waveform of integers to an already existing waveform of doubles.

Parameters:

iw A pointer to an existing waveform of integers *w* A pointer to the waveform of integers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 220 of file intwf.c.

References bpm_error(), intwf_t::ns, intwf_t::wf, and doublewf_t::wf.

6.11.9.43 EXTERN int intwf_compat (intwf_t * *w1*, intwf_t * *w2*)

Checks compatiblity of the two waveforms, returns true if the number of samples and the sampling frequencies match. For the sampling frequency, it is simply checked whether they match to WF_EPS.

Parameters:

w1 A pointer to the first waveform of integers

w2 A pointer to the second waveform of integers

Returns:

1 if the waveforms match, 0 if not.

Definition at line 238 of file intwf.c.

References bpm_error(), intwf_t::fs, intwf_t::ns, and WF_EPS.

Referenced by intwf_add(), intwf_copy(), intwf_divide(), intwf_multiply(), and intwf_subtract().

6.11.9.44 EXTERN int intwf_add (intwf_t * *w1*, intwf_t * *w2*)

Adds two waveforms of integers w1+w2 sample per sample. The result is stored in w1.

Parameters:

w1 A pointer to the first waveform of integers

w2 A pointer to the second waveform of integers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 251 of file intwf.c.

References bpm_error(), bpm_warning(), intwf_compat(), intwf_t::ns, and intwf_t::wf.

6.11.9.45 EXTERN int intwf_subtract (intwf_t * *w1*, intwf_t * *w2*)

Subtracts two waveforms of integers w1-w2 sample per sample. The result is stored in w1.

Parameters:

w1 A pointer to the first waveform of integers

w2 A pointer to the second waveform of integers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 271 of file intwf.c.

References bpm_error(), bpm_warning(), intwf_compat(), intwf_t::ns, and intwf_t::wf.

6.11.9.46 EXTERN int intwf_multiply (intwf_t * *w1*, intwf_t * *w2*)

Multiplies two waveforms of integers w1*w2 sample per sample. The result is stored in w1.

Parameters:

- ***w1*** A pointer to the first waveform of integers
- ***w2*** A pointer to the second waveform of integers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 291 of file intwf.c.

References bpm_error(), bpm_warning(), intwf_compat(), intwf_t::ns, and intwf_t::wf.

6.11.9.47 EXTERN int intwf_divide (intwf_t * *w1*, intwf_t * *w2*)

Divides two waveforms of integers w1/w2 sample per sample. The result is stored in w1. When w2[i] is 0, w1[i] will be set to 0. and a warning message is printed.

Parameters:

- ***w1*** A pointer to the first waveform of integers
- ***w2*** A pointer to the second waveform of integers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 313 of file intwf.c.

References bpm_error(), bpm_warning(), intwf_compat(), intwf_t::ns, and intwf_t::wf.

6.11.9.48 EXTERN int intwf_scale (int *f*, intwf_t * *w*)

Scales the waveform of integers w by factor f. The result is stored in w.

Parameters:

- ***f*** The scalefactor
- ***w*** A pointer to the waveform of integers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 343 of file intwf.c.

References bpm_error(), intwf_t::ns, and intwf_t::wf.

6.11.9.49 EXTERN int intwf_bias (int *c*, intwf_t * *w*)

Biases the waveform of integers w by a constant c. The result is stored in w.

Parameters:

c The constant bias.

w A pointer to the waveform of integers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 360 of file intwf.c.

References bpm_error(), intwf_t::ns, and intwf_t::wf.

6.11.9.50 EXTERN int intwf_add_cwtone (intwf_t * *w*, double *amp*, double *phase*, double *freq*, double *phasenoise*)

Adds a cosine-like CW tone to the entire waveform. The sampling time is taken on the array index, so t=(double)i/w->fs.

Parameters:

w A pointer to the waveform structure

amp Amplitude of the CW tone

phase Phase of the CW tone

freq Frequency of the CW tone

phasenoise Sigma of the gaussian phasenoise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 377 of file intwf.c.

References bpm_error(), dround(), intwf_t::fs, nr_rangauss(), intwf_t::ns, and intwf_t::wf.

6.11.9.51 EXTERN int intwf_add_dcywave (intwf_t * *w*, double *amp*, double *phase*, double *freq*, double *ttrig*, double *tdcy*, double *phasenoise*)

Adds a decaying wave pulse to the waveform. The sampling time is taken on the array index, so t=(double)i/w->fs. The added signal is of the form :

$$ampe^{-(t-ttrig)/tdcy}cos(2\pi freq(t-ttrig)+phase)$$

If desired, phasenoise is added to the phase of the waveform.

Parameters:

w A pointer to the waveform structure

amp Amplitude of the CW tone

phase Phase of the CW tone

freq Frequency of the CW tone

ttrig Trigger time of the pulse

tdcy Decay time of the pulse

phasenoise Sigma of the gaussian phasenoise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 397 of file intwf.c.

References bpm_error(), dround(), intwf_t::fs, nr_rangauss(), intwf_t::ns, and intwf_t::wf.

6.11.9.52 EXTERN int intwf_add_ampnoise (intwf_t * *w*, double *sigma*)

Adds gaussian amplitude noise to the waveform.

Parameters:

w A pointer to the waveform structure

sigma The gaussian sigma of the amplitude noise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 423 of file intwf.c.

References bpm_error(), dround(), nr_rangauss(), intwf_t::ns, and intwf_t::wf.

Referenced by digitise().

6.11.9.53 EXTERN int intwf_basic_stats (intwf_t * *w*, int *s0*, int *s1*, wfstat_t * *stats*)

Retrieves some basic statistics about the waveform of integers in w, only considers samples between s0 and s1.

Parameters:

w A pointer to the waveform structure

s0 First sample to consider

s1 Last sample to consider

stats A filled **wfstat_t** (p. 153) structure is returned.

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 443 of file intwf.c.

References bpm_error(), doublewf_basic_stats(), doublewf_cast_new(), and doublewf_delete().

6.11.9.54 EXTERN int intwf_derive (intwf_t * *w*)

Produce the derivative waveform for w : dw/dt.

Parameters:

w A pointer to the waveform structure.

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 469 of file intwf.c.

References bpm_error(), dround(), intwf_t::fs, intwf_t::ns, and intwf_t::wf.

6.11.9.55 EXTERN int intwf_integrate (intwf_t ∗ *w*)

Produce the integrated waveform for w : ^t w(s)ds.

Parameters:

w A pointer to the waveform structure.

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure

Definition at line 494 of file intwf.c.

References bpm_error(), dround(), intwf_t::fs, intwf_t::ns, and intwf_t::wf.

6.11.9.56 EXTERN void intwf_print (FILE ∗ *of*, intwf_t ∗ *w*)

Print the waveform to the filepointer

Parameters:

of A filepointer, use stdout for the terminal
w A pointer to the waveform

Returns:

void

Definition at line 525 of file intwf.c.

References bpm_error(), intwf_t::fs, intwf_t::ns, and intwf_t::wf.

6.11.9.57 EXTERN int intwf_getvalue (intwf_t ∗ *w*, double *t*, unsigned int *mode*)

Return the value for the waveform at sample time t, according to the interpolation mode.

Parameters:

w A pointer to the waveform structure
t A time at which to sample the waveform
mode Interpolation mode

Returns:

the value of the waveform at time t

Definition at line 544 of file intwf.c.

References bpm_error(), doublewf_cast_new(), doublewf_delete(), doublewf_getvalue(), and dround().

6.11.9.58 EXTERN int intwf_resample (intwf_t * *w2*, double *fs*, intwf_t * *w1*, unsigned int *mode*)

Resamples the waveform w1 into w2 with new fs sampling frequency This routine recalculates the correct number of samples required. However the user needs to make sure that there are enough samples in w2 available as this is not checked. The w2->ns value will be overwritten with the correct amount. The routine checkes whether the maximum allowed number of samples is not exceeded to avoid memory problems.

Parameters:

w A pointer to the waveform structure

t A time at which to sample the waveform

mode Interpolation mode

Returns:

the value of the waveform at time t

Definition at line 571 of file intwf.c.

References bpm_error(), doublewf_cast_new(), doublewf_delete(), doublewf_getvalue(), dround(), intwf_t::fs, MAX_ALLOWED_NS, intwf_t::ns, and intwf_t::wf.

6.11.9.59 EXTERN complexwf_t* complexwf (int *ns*, double *fs*)

Allocates memory for a new waveform of complex numbers

Parameters:

ns The number of samples in the waveform

fs The sampling frequency of the waveform

Returns:

A pointer to the allocated waveform structure

Definition at line 9 of file complexwf.c.

References bpm_error(), complexwf_t::fs, MAX_ALLOWED_NS, complexwf_t::ns, and complexwf_t::wf.

Referenced by complexwf_copy_new().

6.11.9.60 EXTERN complexwf_t* complexwf_copy_new (complexwf_t * *w*)

Allocates memory and produces a copy of the complex waveform w;

Parameters:

w A pointer to the original waveform

Returns:

A pointer to the copy of w

Definition at line 51 of file complexwf.c.

References bpm_error(), complexwf(), complexwf_t::fs, complexwf_t::ns, and complexwf_t::wf.

6.11.9.61 EXTERN int complexwf_copy (complexwf_t * *copy*, complexwf_t * *src*)

Copies the values from existing complex waveform src into copy checks first whether the waveforms are compatible... This routine doesn't allocate memory internally and the waveforms should already have been created by the user...

Parameters:

- ***copy*** A pointer to the copy waveform
- ***src*** A pointer to the original waveform

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 68 of file complexwf.c.

References bpm_error(), complexwf_compat(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.62 EXTERN int complexwf_subset (complexwf_t * *sub*, complexwf_t * *w*, int *i1*, int *i2*)

Copies a subset from sample i1 to sample i2 (inclusive) to the sub waveform from complex waveform w. The routine expects the sub waveform to already exist with enough samples. (this is not checked !) The sub->fs and sub->ns will be overwritten.

Parameters:

- ***sub*** Pointer to the waveform which will hold the subset
- ***w*** Pointer to the original waveform
- ***i1*** First sample of w to copy
- ***i2*** Last sample of w to copy

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 89 of file complexwf.c.

References bpm_error(), complexwf_t::fs, complexwf_t::ns, and complexwf_t::wf.

6.11.9.63 EXTERN int complexwf_setvalues (complexwf_t * *w*, complex_t * *x*)

Fills the complex waveform with the values from the array x. No check is performed whether x contains enough samples, the user needs to be sure this is the case !

Parameters:

- ***w*** A pointer to the waveform of complex numbers
- ***x*** A pointer to the complex x values

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 113 of file complexwf.c.

References bpm_error(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.64 EXTERN int complexwf_setfunction (complexwf_t ∗ *w*, complex_t(∗)(double, int, double ∗) *wffun*, int *npars*, double ∗ *par*)

Fills the waveform with values from the function wffun(), this function has to return a **complex_t** (p. 140) from argument t (time) and has npars parameters given by the array ∗par. The function will be evaluated at the time t of each sample...

Parameters:

w A pointer to the waveform of complex numbers

wffun A pointer to the function to fill the waveform with

t The time parameter in the function

npars Number of parameters for the function

par Array of parameters for the function

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 128 of file complexwf.c.

References bpm_error(), complexwf_t::fs, complexwf_t::ns, and complexwf_t::wf.

6.11.9.65 EXTERN int complexwf_reset (complexwf_t ∗ *w*)

Resets the waveform of complex numbers to 0+0i

Parameters:

w A pointer to the complex waveform

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 146 of file complexwf.c.

References bpm_error(), complexwf_t::ns, and complexwf_t::wf.

Referenced by get_mode_response().

6.11.9.66 EXTERN void complexwf_delete (complexwf_t ∗ *w*)

Frees up the memory used by the waveform

Parameters:

w A pointer to the waveform of complex numbers

Returns:

void

Definition at line 163 of file complexwf.c.

References bpm_warning(), and complexwf_t::wf.

6.11.9.67 EXTERN int complexwf_compat (complexwf_t * *w1*, complexwf_t * *w2*)

Checks compatiblity of the two waveforms, returns true if the number of samples and the sampling frequencies match. For the sampling frequency, it is simply checked whether they match to WF_EPS.

Parameters:

w1 A pointer to the first waveform of complex numbers

w2 A pointer to the second waveform of complex numbers

Returns:

1 if the waveforms match, 0 if not.

Definition at line 180 of file complexwf.c.

References bpm_error(), complexwf_t::fs, complexwf_t::ns, and WF_EPS.

Referenced by complexwf_add(), complexwf_copy(), complexwf_divide(), complexwf_multiply(), and complexwf_subtract().

6.11.9.68 EXTERN int complexwf_add (complexwf_t * *w1*, complexwf_t * *w2*)

Adds two waveforms of complex numbers w1+w2 sample per sample. The result is stored in w1.

Parameters:

w1 A pointer to the first waveform of complex numbers

w2 A pointer to the second waveform of comlex numbers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 193 of file complexwf.c.

References bpm_error(), bpm_warning(), complexwf_compat(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.69 EXTERN int complexwf_subtract (complexwf_t * *w1*, complexwf_t * *w2*)

Subtracts two waveforms of complex numbers w1-w2 sample per sample. The result is stored in w1.

Parameters:

w1 A pointer to the first waveform of complex numbers

w2 A pointer to the second waveform of comlex numbers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 213 of file complexwf.c.

References bpm_error(), bpm_warning(), complexwf_compat(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.70 EXTERN int complexwf_multiply (complexwf_t * *w1*, complexwf_t * *w2*)

Multiplies two waveforms of complex numbers w1*w2 sample per sample. The result is stored in w1.

Parameters:

w1 A pointer to the first waveform of complex numbers

w2 A pointer to the second waveform of comlex numbers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 234 of file complexwf.c.

References bpm_error(), bpm_warning(), complexwf_compat(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.71 EXTERN int complexwf_divide (complexwf_t * *w1*, complexwf_t * *w2*)

Divides two waveforms of complex numbers w1/w2 sample per sample. The result is stored in w1.

Parameters:

w1 A pointer to the first waveform of complex numbers

w2 A pointer to the second waveform of comlex numbers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 256 of file complexwf.c.

References bpm_error(), bpm_warning(), complexwf_compat(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.72 EXTERN int complexwf_scale (complex_t *f*, complexwf_t * *w*)

Scales the waveform of complex numbers w with complex factor f The result is stored in w.

Parameters:

f The complex scaling factor

w A pointer to the waveform of comlex numbers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 288 of file complexwf.c.

References bpm_error(), complexwf_t::ns, and complexwf_t::wf.

Referenced by rf_amplify_complex(), and rf_phase_shifter().

6.11.9.73 EXTERN int complexwf_bias (complex_t *c*, complexwf_t * *w*)

Biases the waveform of complex numbers w with complex constant c The result is stored in w.

Parameters:

c The complex constant

w A pointer to the waveform of comlex numbers

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 305 of file complexwf.c.

References bpm_error(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.74 EXTERN int complexwf_add_cwtone (complexwf_t * *w*, double *amp*, double *phase*, double *freq*, double *phasenoise*)

Adds a CW tone to the entire waveform. The sampling time is taken on the array index, so t=(double)i/w->fs. The real part will have the cos-like waveform, the imaginary part the sin-like waveform.

Parameters:

w A pointer to the complex waveform structure

amp Amplitude of the CW tone

phase Phase of the CW tone

freq Frequency of the CW tone

phasenoise Sigma of the gaussian phasenoise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 322 of file complexwf.c.

References bpm_error(), complexwf_t::fs, complex_t::im, nr_rangauss(), complexwf_t::ns, complex_t::re, and complexwf_t::wf.

6.11.9.75 EXTERN int complexwf_add_dcywave (complexwf_t * *w*, double *amp*, double *phase*, double *freq*, double *ttrig*, double *tdcy*, double *phasenoise*)

Adds a decaying wave pulse to the waveform. The sampling time is taken on the array index, so t=(double)i/w->fs. The added signal is of the form :

$$ampe^{-(t-ttrig)/tdcy}sin(2\pi freq(t-ttrig)+phase)$$

The real part will have the cos-like component, the imaginary part the sin-like component. If desired, phasenoise is added to the phase of the waveform.

Parameters:

w A pointer to the waveform structure

amp Amplitude of the CW tone

phase Phase of the CW tone

freq Frequency of the CW tone

ttrig Trigger time of the pulse

tdcy Decay time of the pulse

phasenoise Sigma of the gaussian phasenoise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 346 of file complexwf.c.

References bpm_error(), complexwf_t::fs, complex_t::im, nr_rangauss(), complexwf_t::ns, complex_t::re, and complexwf_t::wf.

6.11.9.76 EXTERN int complexwf_add_noise (complexwf_t ∗ *w*, double *sigma*)

Adds uncorrelated gaussian amplitude noise with uniformly distributed random phase to the complex the waveform.

Parameters:

w A pointer to the complex waveform structure

sigma The gaussian sigma of the amplitude noise, phase is uniform over 2pi

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 373 of file complexwf.c.

References bpm_error(), nr_rangauss(), nr_ranuniform(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.77 EXTERN int complexwf_add_ampnoise (complexwf_t ∗ *w*, double *sigma*)

Adds pure gaussian amplitude noise to the complex waveform and leaves the phase untouched

Parameters:

w A pointer to the complex waveform structure

sigma The gaussian sigma of the amplitude noise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 397 of file complexwf.c.

References bpm_error(), nr_rangauss(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.78 EXTERN int complexwf_add_phasenoise (complexwf_t ∗ *w*, double *sigma*)

Adds pure gaussian phase noise to the complex waveform and leaves the amplitude untouched

Parameters:

w A pointer to the complex waveform structure

sigma The gaussian sigma of the phase noise

Returns:

BPM_SUCCESS upon succes, BPM_FAILURE upon failure.

Definition at line 421 of file complexwf.c.

References bpm_error(), nr_rangauss(), complexwf_t::ns, and complexwf_t::wf.

6.11.9.79 EXTERN void complexwf_print (FILE * *of*, complexwf_t * *w*)

Print the waveform to the filepointer

Parameters:

of A filepointer, use stdout for the terminal

w A pointer to the waveform

Returns:

void

Definition at line 446 of file complexwf.c.

References bpm_error(), complexwf_t::fs, complex_t::im, complexwf_t::ns, complex_t::re, and complexwf_t::wf.

6.11.9.80 EXTERN int complexwf_getreal (doublewf_t * *re*, complexwf_t * *z*)

Gets the real part of the comlex waveform into the waveform of doubles. The doublewf needs to be allocated by the user beforehand and have the same number of samples as the complex waveform.

Parameters:

re A pointer to the waveform of doubles which will store the real part

z A pointer to the complex waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 466 of file complexwf.c.

References bpm_error(), bpm_warning(), complexwf_t::ns, doublewf_t::ns, complex_t::re, complexwf_t::wf, and doublewf_t::wf.

Referenced by rf_rectify().

6.11.9.81 EXTERN int complexwf_getimag (doublewf_t * *im*, complexwf_t * *z*)

Gets the imaginary part of the comlex waveform into the waveform of doubles. The doublewf needs to be allocated by the user beforehand and have the same number of samples as the complex waveform.

Parameters:

im A pointer to the waveform of doubles which will store the imaginary part

z A pointer to the complex waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 488 of file complexwf.c.

References bpm_error(), bpm_warning(), complex_t::im, complexwf_t::ns, doublewf_t::ns, complexwf_-t::wf, and doublewf_t::wf.

6.11.9.82 EXTERN int complexwf_getamp (doublewf_t ∗ *r*, complexwf_t ∗ *z*)

Gets the amplitude of the comlex waveform into the waveform of doubles. The doublewf needs to be allocated by the user beforehand and have the same number of samples as the complex waveform.

Parameters:

im A pointer to the waveform of doubles which will store the amplitude

z A pointer to the complex waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 510 of file complexwf.c.

References bpm_error(), bpm_warning(), complexwf_t::ns, doublewf_t::ns, complexwf_t::wf, and doublewf_t::wf.

6.11.9.83 EXTERN int complexwf_getphase (doublewf_t ∗ *theta*, complexwf_t ∗ *z*)

Gets the phase of the comlex waveform into the waveform of doubles. The doublewf needs to be allocated by the user beforehand and have the same number of samples as the complex waveform. The phase is normalised between [0,2pi[.

Parameters:

im A pointer to the waveform of doubles which will store the phase

z A pointer to the complex waveform

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 532 of file complexwf.c.

References bpm_error(), bpm_warning(), norm_phase(), complexwf_t::ns, doublewf_t::ns, complexwf_-t::wf, and doublewf_t::wf.

6.11.9.84 EXTERN doublewf_t∗ complexwf_getreal_new (complexwf_t ∗ *z*)

Retrieves the real part of the complex waveform in a newly allocated waveform of doubles. Memory on the heap is allocated inside this routine, the user has to deal with deal with freeing it him/her self.

Parameters:

z A pointer to the complex waveform

Returns:

A pointer to the allocated waveform of doubles containing the real part of z.

Definition at line 601 of file complexwf.c.

References bpm_error(), doublewf(), complexwf_t::fs, complexwf_t::ns, complex_t::re, complexwf_t::wf, and doublewf_t::wf.

6.11.9.85 EXTERN doublewf_t* complexwf_getimag_new (complexwf_t * *z*)

Retrieves the imaginary part of the complex waveform in a newly allocated waveform of doubles. Memory on the heap is allocated inside this routine, the user has to deal with deal with freeing it him/her self.

Parameters:

z A pointer to the complex waveform

Returns:

A pointer to the allocated waveform of doubles containing the imaginary part of z.

Definition at line 626 of file complexwf.c.

References bpm_error(), doublewf(), complexwf_t::fs, complex_t::im, complexwf_t::ns, complexwf_t::wf, and doublewf_t::wf.

6.11.9.86 EXTERN doublewf_t* complexwf_getamp_new (complexwf_t * *z*)

Retrieves the amplitude of the complex waveform in a newly allocated waveform of doubles. Memory on the heap is allocated inside this routine, the user has to deal with deal with freeing it him/her self.

Parameters:

z A pointer to the complex waveform

Returns:

A pointer to the allocated waveform of doubles containing the amplitude of z.

Definition at line 651 of file complexwf.c.

References bpm_error(), doublewf(), complexwf_t::fs, complexwf_t::ns, complexwf_t::wf, and doublewf_t::wf.

6.11.9.87 EXTERN doublewf_t* complexwf_getphase_new (complexwf_t * *z*)

Retrieves the phase of the complex waveform in a newly allocated waveform of doubles. Memory on the heap is allocated inside this routine, the user has to deal with deal with freeing it him/her self. The phase is normalised between [0,2pi[.

Parameters:

z A pointer to the complex waveform

Returns:

A pointer to the allocated waveform of doubles containing the phase of z.

Definition at line 676 of file complexwf.c.

References bpm_error(), doublewf(), complexwf_t::fs, norm_phase(), complexwf_t::ns, complexwf_t::wf, and doublewf_t::wf.

6.11.9.88 EXTERN int complexwf_setreal (complexwf_t * *z*, doublewf_t * *re*)

Set the real part of the complex waveform z to re. The complexwf needs to be allocated by the user beforehand and have the same number of samples as the double waveform.

Parameters:

z A pointer to the complex waveform

re A pointer to a waveform of double containing the real part

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 557 of file complexwf.c.

References bpm_error(), bpm_warning(), complexwf_t::ns, doublewf_t::ns, complex_t::re, doublewf_t::wf, and complexwf_t::wf.

Referenced by ddc(), and get_mode_response().

6.11.9.89 EXTERN int complexwf_setimag (complexwf_t * *z*, doublewf_t * *im*)

Set the imaginary part of the complex waveform z to im. The complexwf needs to be allocated by the user beforehand and have the same number of samples as the double waveform.

Parameters:

z A pointer to the complex waveform

re A pointer to a waveform of double containing the imaginary part

Returns:

BPM_SUCCESS upon success, BPM_FAILURE upon failure

Definition at line 579 of file complexwf.c.

References bpm_error(), bpm_warning(), complex_t::im, complexwf_t::ns, doublewf_t::ns, doublewf_t::wf, and complexwf_t::wf.

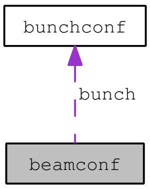
Referenced by ddc(), and get_mode_response().

7 Data Structure Documentation

7.1 beamconf Struct Reference

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

Collaboration diagram for beamconf:



7.1.1 Detailed Description

This structure contains the global beam parameters as well as a pointer to the array of bunches

Definition at line 227 of file bpm_interface.h.

Data Fields

- int **train_num**
- double **beamrate**
- double **bunchrate**
- int **nbunches**
- **bunchconf_t** $*$ **bunch**
- double **position** [2]
- double **positionsigma** [2]
- double **slope** [2]
- double **slopesigma** [2]
- double **tilt** [2]
- double **tiltsigma** [2]
- double **bunchlength**
- double **bunchlengthsigma**
- double **energy**
- double **energysigma**
- double **charge**
- double **chargesigma**

7.1.2 Field Documentation

7.1.2.1 int beamconf::train_num

seq number of the train (evt num)

Definition at line 228 of file bpm_interface.h.

7.1.2.2 double beamconf::beamrate

beam repetition rate (train to train)

Definition at line 230 of file bpm_interface.h.

7.1.2.3 double beamconf::bunchrate

bunch repetition rate (in the train)

Definition at line 231 of file bpm_interface.h.

7.1.2.4 int beamconf::nbunches

number of bunches per train

Definition at line 232 of file bpm_interface.h.

Referenced by generate_bpmsignal(), and get_bpmhits().

7.1.2.5 bunchconf_t* beamconf::bunch

list of pointers to the bunch conf structures

Definition at line 234 of file bpm_interface.h.

Referenced by generate_bpmsignal(), and get_bpmhits().

7.1.2.6 double beamconf::position[2]

beam position at the origin

Definition at line 236 of file bpm_interface.h.

7.1.2.7 double beamconf::positionsigma[2]

position spread at the origin

Definition at line 237 of file bpm_interface.h.

7.1.2.8 double beamconf::slope[2]

beam slope at the origin

Definition at line 239 of file bpm_interface.h.

7.1.2.9 double beamconf::slopesigma[2]

slope spread at the origin

Definition at line 240 of file bpm_interface.h.

7.1.2.10 double beamconf::tilt[2]

bunch tilt at the origin

Definition at line 242 of file bpm_interface.h.

7.1.2.11 double beamconf::tiltsigma[2]

tilt spread at the origin

Definition at line 243 of file bpm_interface.h.

7.1.2.12 double beamconf::bunchlength

bunch length at the origin

Definition at line 245 of file bpm_interface.h.

7.1.2.13 double beamconf::bunchlengthsigma

length spread at the origin

Definition at line 246 of file bpm_interface.h.

7.1.2.14 double beamconf::energy

beam energy (in GeV) at the origin

Definition at line 248 of file bpm_interface.h.

7.1.2.15 double beamconf::energysigma

beam energy spread

Definition at line 249 of file bpm_interface.h.

7.1.2.16 double beamconf::charge

bunch charge (in nC)

Definition at line 250 of file bpm_interface.h.

7.1.2.17 double beamconf::chargesigma

charge spread

Definition at line 251 of file bpm_interface.h.

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

- bpminterface/**bpm_interface.h**

7.2 bpmcalib Struct Reference

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

7.2.1 Detailed Description

A structure containing the calibration information : purely calibration !

Definition at line 152 of file bpm_interface.h.

Data Fields

- double **ddc_IQphase**
- double **ddc_posscale**
- double **ddc_slopescale**

- double **ddc_ct_amp**
- double **ddc_ct_phase**
- double **fit_IQphase**
- double **fit_posscale**
- double **fit_slopescale**
- double **fit_ct_amp**
- double **fit_ct_phase**

7.2.2 Field Documentation

7.2.2.1 double bpmcalib::ddc_IQphase

processed IQ phase for the ddc routine

Definition at line 154 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.2.2.2 double bpmcalib::ddc_posscale

processed position scale for the ddc routine

Definition at line 155 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.2.2.3 double bpmcalib::ddc_slopescale

processed slope scale for the fit routine

Definition at line 156 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.2.2.4 double bpmcalib::ddc_ct_amp

calibration tone amplitude at time of calibration

Definition at line 157 of file bpm_interface.h.

Referenced by correct_gain().

7.2.2.5 double bpmcalib::ddc_ct_phase

calibration tone phase at time of calibration

Definition at line 158 of file bpm_interface.h.

Referenced by correct_gain().

7.2.2.6 double bpmcalib::fit_IQphase

processed IQ phase for the fit routine

Definition at line 161 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.2.2.7 double bpmcalib::fit_posscale

position scale for the fit routine

Definition at line 162 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.2.2.8 double bpmcalib::fit_slopescale

slope scale for the fit routine

Definition at line 163 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.2.2.9 double bpmcalib::fit_ct_amp

calibration tone amplitude at time of calibration

Definition at line 164 of file bpm_interface.h.

Referenced by correct_gain().

7.2.2.10 double bpmcalib::fit_ct_phase

calibration tone phase at time of calibration

Definition at line 165 of file bpm_interface.h.

Referenced by correct_gain().

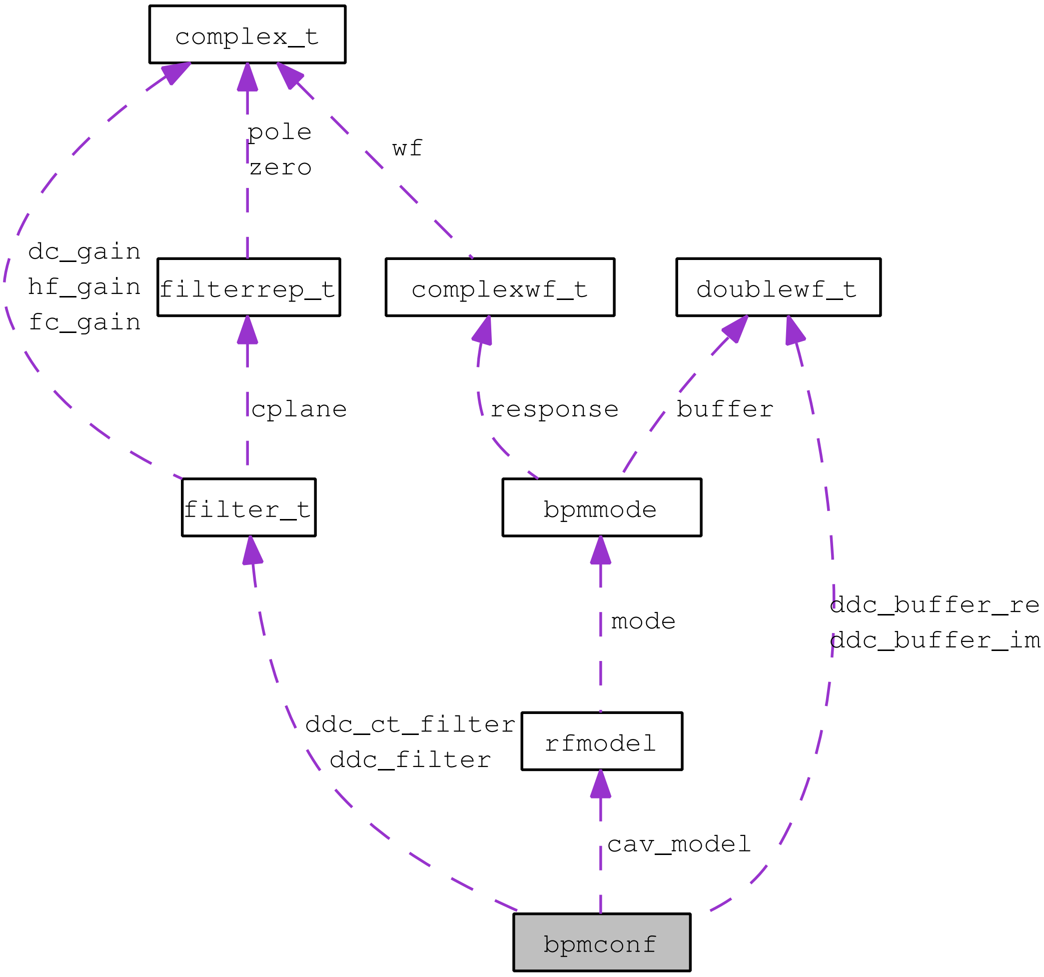
The documentation for this struct was generated from the following file:

- bpminterface/**bpm_interface.h**

7.3 bpmconf Struct Reference

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

Collaboration diagram for bpmconf:



7.3.1 Detailed Description

Structure containing the BPM configuration

Definition at line 86 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**
- **rfmodel_t** ∗ **cav_model**
- double **cav_length**
- double **cav_freq**
- double **cav_decaytime**
- double **cav_phase**

- double **cav_iqrotation**
- double **cav_chargesens**
- double **cav_possens**
- double **cav_tiltsens**
- double **rf_LOfreq**
- double **digi_trigtimeoffset**
- double **digi_freq**
- int **digi_nbits**
- int **digi_nsamples**
- double **digi_ampnoise**
- int **digi_voltageoffset**
- double **digi_phasenoise**
- double **t0**
- double **ddc_freq**
- double **ddc_tdecay**
- double **ddc_tOffset**
- **filter_t** ∗ **ddc_filter**
- double **fit_inifreq**
- double **fit_initdecay**
- double **fit_tOffset**
- double **ddc_ct_freq**
- **filter_t** ∗ **ddc_ct_filter**
- int **ddc_ct_iSample**
- double **geom_pos** [3]
- double **geom_tilt** [3]
- int **ref_idx**
- int **diode_idx**
- int **forced_trigger**
- **doublewf_t** ∗ **ddc_buffer_re**
- **doublewf_t** ∗ **ddc_buffer_im**

7.3.2 Field Documentation

7.3.2.1 char bpmconf::name[20]

a BPM should have a name

Definition at line 87 of file bpm_interface.h.

Referenced by postprocess_waveform(), process_caltone(), process_diode(), process_dipole(), process_-monopole(), and process_waveform().

7.3.2.2 enum bpmtype_t bpmconf::cav_type

BPM type

Definition at line 89 of file bpm_interface.h.

Referenced by process_diode().

7.3.2.3 enum bpmpol_t bpmconf::cav_polarisation

BPM polarisation

Definition at line 90 of file bpm_interface.h.

7.3.2.4 enum bpmphase_t bpmconf::cav_phasetype

BPM phase type

Definition at line 91 of file bpm_interface.h.

7.3.2.5 double bpmconf::cav_length

length of the cavity

Definition at line 94 of file bpm_interface.h.

Referenced by get_mode_amplitude().

7.3.2.6 double bpmconf::cav_freq

cavity freq (MHz)

Definition at line 95 of file bpm_interface.h.

7.3.2.7 double bpmconf::cav_decaytime

cavity decay time (microsec)

Definition at line 96 of file bpm_interface.h.

Referenced by process_waveform().

7.3.2.8 double bpmconf::cav_phase

phase advance wrt. reference (fixed or random)

Definition at line 97 of file bpm_interface.h.

7.3.2.9 double bpmconf::cav_iqrotation

cavity IQ rotation

Definition at line 98 of file bpm_interface.h.

7.3.2.10 double bpmconf::cav_chargesens

charge sensitivity (volt/nC)

Definition at line 99 of file bpm_interface.h.

7.3.2.11 double bpmconf::cav_possens

pos sensitivity at 1.6nC charge (volt/micron)

Definition at line 100 of file bpm_interface.h.

7.3.2.12 double bpmconf::cav_tiltsens

tilt sensitivity at 1.6nC charge (volt/micron)

Definition at line 101 of file bpm_interface.h.

7.3.2.13 double bpmconf::rf_LOfreq

LO frequency to mix down with (in MHz)

Definition at line 103 of file bpm_interface.h.

7.3.2.14 double bpmconf::digi_trigtimeoffset

time (usec) to offset bunch arrival times by

Definition at line 106 of file bpm_interface.h.

7.3.2.15 double bpmconf::digi_freq

digitization frequency (MHz)

Definition at line 107 of file bpm_interface.h.

Referenced by process_waveform().

7.3.2.16 int bpmconf::digi_nbits

number of bits in ADC for digitisation

Definition at line 108 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.3.2.17 int bpmconf::digi_nsamples

number of samples in ADC digitisation

Definition at line 109 of file bpm_interface.h.

Referenced by process_waveform().

7.3.2.18 double bpmconf::digi_ampnoise

amplitude noise in ADC channels (pedestal width)

Definition at line 110 of file bpm_interface.h.

7.3.2.19 int bpmconf::digi_voltageoffset

voltage offset (pedestal position) in counts

Definition at line 111 of file bpm_interface.h.

7.3.2.20 double bpmconf::digi_phasenoise

phase noise

Definition at line 112 of file bpm_interface.h.

7.3.2.21 double bpmconf::t0

start time of pulse

Definition at line 116 of file bpm_interface.h.

Referenced by process_waveform().

7.3.2.22 double bpmconf::ddc_freq

Frequency of downmixed waveform (MHz)

Definition at line 119 of file bpm_interface.h.

Referenced by process_waveform().

7.3.2.23 double bpmconf::ddc_tdecay

Decay time (usec)

Definition at line 120 of file bpm_interface.h.

Referenced by process_waveform().

7.3.2.24 double bpmconf::ddc_tOffset

Always have offset from t0 for sampling !!!

Definition at line 121 of file bpm_interface.h.

Referenced by process_waveform().

7.3.2.25 filter_t* bpmconf::ddc_filter

DDC 2 omega filter

Definition at line 122 of file bpm_interface.h.

Referenced by process_waveform().

7.3.2.26 double bpmconf::fit_inifreq

Initial frequency for fitting

Definition at line 125 of file bpm_interface.h.

7.3.2.27 double bpmconf::fit_initdecay

Initial decay time for fitting

Definition at line 126 of file bpm_interface.h.

7.3.2.28 double bpmconf::fit_tOffset

Offset from t0 to start fitting

Definition at line 127 of file bpm_interface.h.

Referenced by process_waveform().

7.3.2.29 double bpmconf::ddc_ct_freq

caltone frequency for the ddc algorithm

Definition at line 130 of file bpm_interface.h.

Referenced by process_caltone().

7.3.2.30 filter_t* bpmconf::ddc_ct_filter

filter for the caltone ddc

Definition at line 131 of file bpm_interface.h.

Referenced by process_caltone().

7.3.2.31 int bpmconf::ddc_ct_iSample

sample number to sample from ddc for amp/phase

Definition at line 132 of file bpm_interface.h.

Referenced by process_caltone().

7.3.2.32 double bpmconf::geom_pos[3]

position of the BPM in the beamline

Definition at line 136 of file bpm_interface.h.

Referenced by get_bpmhit().

7.3.2.33 double bpmconf::geom_tilt[3]

tilt of the BPM (0: xrot, 1: yrot, 2: zrot)

Definition at line 137 of file bpm_interface.h.

Referenced by get_bpmhit().

7.3.2.34 int bpmconf::ref_idx

reference cavity index for this BPM

Definition at line 140 of file bpm_interface.h.

7.3.2.35 int bpmconf::diode_idx

reference diode index for this BPM

Definition at line 141 of file bpm_interface.h.

7.3.2.36 int bpmconf::forced_trigger

this cavity is abused as trigger signal

Definition at line 142 of file bpm_interface.h.

Referenced by process_diode().

7.3.2.37 doublewf_t* bpmconf::ddc_buffer_re

pointer to a **doublewf_t** (p. 141) buffer

Definition at line 145 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.3.2.38 doublewf_t∗ bpmconf::ddc_buffer_im

pointer to a **doublewf_t** (p. 141) buffer

Definition at line 146 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

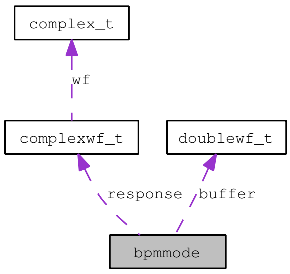
The documentation for this struct was generated from the following file:

- bpminterface/**bpm_interface.h**

7.4 bpmmode Struct Reference

`#include <bpm_interface.h>`

Collaboration diagram for bpmmode:



7.4.1 Detailed Description

This structure defines a BPM resonant mode which is defined by it's resonant frequency, Q factor and sensitivities to the beam charge, slope and bunch tilt.

Definition at line 282 of file bpm_interface.h.

Data Fields

- char **name** [20]
- double **frequency**
- double **Q**
- int **order**
- enum **bpmpol_t polarisation**
- double **sensitivity**
- **complexwf_t** ∗ **response**
- **doublewf_t** ∗ **buffer**

7.4.2 Field Documentation

7.4.2.1 char bpmmode::name[20]

The name for the BPM mode, e.g "dipolex"

Definition at line 283 of file bpm_interface.h.

Referenced by generate_bpmsignal().

7.4.2.2 double bpmmode::frequency

The resonant frequency of the mode

Definition at line 284 of file bpm_interface.h.

Referenced by get_mode_amplitude(), and get_mode_response().

7.4.2.3 double bpmmode::Q

The Q factor for the mode

Definition at line 285 of file bpm_interface.h.

Referenced by get_mode_response().

7.4.2.4 int bpmmode::order

The mode order, 0:monopole, 1:dipole, 2:quadrupole...

Definition at line 286 of file bpm_interface.h.

Referenced by add_mode_response(), get_mode_amplitude(), and get_mode_response().

7.4.2.5 enum bpmpol_t bpmmode::polarisation

The mode polarisation: horiz, vert

Definition at line 287 of file bpm_interface.h.

Referenced by get_mode_amplitude().

7.4.2.6 double bpmmode::sensitivity

The sensitivity of the mode, units depend on order

Definition at line 288 of file bpm_interface.h.

Referenced by get_mode_amplitude().

7.4.2.7 complexwf_t∗ bpmmode::response

Pointer to the mode response buffer

Definition at line 289 of file bpm_interface.h.

Referenced by add_mode_response(), generate_bpmsignal(), and get_mode_response().

7.4.2.8 doublewf_t∗ bpmmode::buffer

Pointer to the mode's buffer

Definition at line 290 of file bpm_interface.h.

Referenced by generate_bpmsignal().

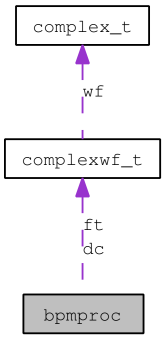
The documentation for this struct was generated from the following file:

- bpminterface/**bpm_interface.h**

7.5 bpmproc Struct Reference

`#include <bpm_interface.h>`

Collaboration diagram for bpmproc:



7.5.1 Detailed Description

A structure containing the processed waveform information

Definition at line 171 of file bpm_interface.h.

Data Fields

- double **ampnoise**
- double **voltageoffset**
- double **t0**
- int **saturated**
- int **iunsat**
- **complexwf_t** ∗ **dc**
- **complexwf_t** ∗ **ft**
- int **fft_success**
- double **fft_amp**
- double **fft_freq**
- double **fft_tdecay**
- double **fft_offset**
- int **ddc_success**
- double **ddc_tSample**
- int **ddc_iSample**
- double **ddc_Q**
- double **ddc_I**
- double **ddc_amp**
- double **ddc_phase**
- double **ddc_tdecay**
- double **ddc_pos**
- double **ddc_slope**

- double **ddc_ct_amp**
- double **ddc_ct_phase**
- int **fit_success**
- double **fit_Q**
- double **fit_I**
- double **fit_amp**
- double **fit_phase**
- double **fit_freq**
- double **fit_tdecay**
- double **fit_offset**
- double **fit_pos**
- double **fit_slope**
- double **fit_ct_amp**
- double **fit_ct_phase**

7.5.2 Field Documentation

7.5.2.1 double bpmproc::ampnoise

calculated (processed) amplitude noise

Definition at line 172 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.5.2.2 double bpmproc::voltageoffset

calculated voltage offset

Definition at line 173 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.5.2.3 double bpmproc::t0

trigger t0 for int, copied from **bpmconf_t::t0** (p. 127) for ext

Definition at line 175 of file bpm_interface.h.

Referenced by process_diode(), and process_waveform().

7.5.2.4 int bpmproc::saturated

this signal was saturated

Definition at line 177 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.5.2.5 int bpmproc::iunsat

the last unsaturated sample index

Definition at line 178 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.5.2.6 complexwf_t* bpmproc::dc

The signal's DC waveform

Definition at line 180 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.5.2.7 complexwf_t* bpmproc::ft

The signal's fourier transform

Definition at line 181 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.5.2.8 int bpmproc::fft_success

do we have proper fft info ?

Definition at line 184 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.5.2.9 double bpmproc::fft_amp

amplitude of fft

Definition at line 185 of file bpm_interface.h.

7.5.2.10 double bpmproc::fft_freq

frequency obtained from fft (MHz)

Definition at line 186 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.5.2.11 double bpmproc::fft_tdecay

decay time obtained from fft (usec)

Definition at line 187 of file bpm_interface.h.

Referenced by process_caltone(), and process_waveform().

7.5.2.12 double bpmproc::fft_offset

offset of fft in fit

Definition at line 188 of file bpm_interface.h.

7.5.2.13 int bpmproc::ddc_success

do we have proper ddc info ?

Definition at line 191 of file bpm_interface.h.

Referenced by correct_gain(), postprocess_waveform(), process_caltone(), and process_waveform().

7.5.2.14 double bpmproc::ddc_tSample

time at which the ddc was sampled, t0+t0Offset

Definition at line 192 of file bpm_interface.h.

Referenced by process_waveform().

7.5.2.15 int bpmproc::ddc_iSample

index of sample at which ddc sample was taken

Definition at line 193 of file bpm_interface.h.

Referenced by process_waveform().

7.5.2.16 double bpmproc::ddc_Q

ddc Q value

Definition at line 194 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.5.2.17 double bpmproc::ddc_I

ddc I value

Definition at line 195 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.5.2.18 double bpmproc::ddc_amp

downconverted amplitude

Definition at line 196 of file bpm_interface.h.

Referenced by correct_gain(), postprocess_waveform(), process_caltone(), and process_waveform().

7.5.2.19 double bpmproc::ddc_phase

downconverted phase

Definition at line 197 of file bpm_interface.h.

Referenced by correct_gain(), postprocess_waveform(), process_caltone(), and process_waveform().

7.5.2.20 double bpmproc::ddc_tdecay

downconverted decay time of waveform

Definition at line 198 of file bpm_interface.h.

7.5.2.21 double bpmproc::ddc_pos

calculated position from ddc

Definition at line 200 of file bpm_interface.h.

Referenced by ana_compute_residual(), and postprocess_waveform().

7.5.2.22 double bpmproc::ddc_slope

calculated slope from ddc

Definition at line 201 of file bpm_interface.h.

Referenced by ana_compute_residual(), and postprocess_waveform().

7.5.2.23 double bpmproc::ddc_ct_amp

last measured calibration tone amplitude for this bpm

Definition at line 203 of file bpm_interface.h.

Referenced by correct_gain(), and process_caltone().

7.5.2.24 double bpmproc::ddc_ct_phase

last measured calibration tone phase for this bpm

Definition at line 204 of file bpm_interface.h.

Referenced by correct_gain(), and process_caltone().

7.5.2.25 int bpmproc::fit_success

do we have proper fit info ?

Definition at line 207 of file bpm_interface.h.

Referenced by correct_gain(), postprocess_waveform(), and process_waveform().

7.5.2.26 double bpmproc::fit_Q

fit Q value

Definition at line 208 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.5.2.27 double bpmproc::fit_I

fit I value

Definition at line 209 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.5.2.28 double bpmproc::fit_amp

fitted amplitude

Definition at line 210 of file bpm_interface.h.

Referenced by correct_gain(), postprocess_waveform(), and process_waveform().

7.5.2.29 double bpmproc::fit_phase

fitted phase

Definition at line 211 of file bpm_interface.h.

Referenced by correct_gain(), postprocess_waveform(), and process_waveform().

7.5.2.30 double bpmproc::fit_freq

fitted frequency (MHz)

Definition at line 212 of file bpm_interface.h.

Referenced by process_waveform().

7.5.2.31 double bpmproc::fit_tdecay

fitted decay time of waveform (usec)

Definition at line 213 of file bpm_interface.h.

Referenced by process_waveform().

7.5.2.32 double bpmproc::fit_offset

fitted offset for waveform

Definition at line 214 of file bpm_interface.h.

7.5.2.33 double bpmproc::fit_pos

calculated position from fit

Definition at line 216 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.5.2.34 double bpmproc::fit_slope

calculated slope from fit

Definition at line 217 of file bpm_interface.h.

Referenced by postprocess_waveform().

7.5.2.35 double bpmproc::fit_ct_amp

last measured calibration tone amplitude for this bpm

Definition at line 219 of file bpm_interface.h.

Referenced by correct_gain().

7.5.2.36 double bpmproc::fit_ct_phase

last measured calibration tone phase for this bpm

Definition at line 220 of file bpm_interface.h.

Referenced by correct_gain().

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

- bpminterface/**bpm_interface.h**

7.6 bunchconf Struct Reference

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

7.6.1 Detailed Description

This structure contains information on a single bunch inside the bunchtrain, which has its own energy, internal energy spread, charge, length, position/slope/tilt in the world coo frame and position/slope/tilt in the BPM local coo frame.

Definition at line 260 of file bpm_interface.h.

Data Fields

- int **train_num**
- int **bunch_num**
- double **energy**
- double **energyspread**
- double **charge**
- double **length**
- double **arrival_time**
- double **position** [2]
- double **slope** [2]
- double **tilt** [2]
- double **bpmposition** [3]
- double **bpmslope** [2]
- double **bpmtilt** [2]

7.6.2 Field Documentation

7.6.2.1 int bunchconf::train_num

seq number of the train this bunch belongs to

Definition at line 261 of file bpm_interface.h.

7.6.2.2 int bunchconf::bunch_num

seq number of the bunch in the train

Definition at line 262 of file bpm_interface.h.

7.6.2.3 double bunchconf::energy

energy of the bunch

Definition at line 264 of file bpm_interface.h.

7.6.2.4 double bunchconf::energyspread

energy spread inside the bunch

Definition at line 265 of file bpm_interface.h.

7.6.2.5 double bunchconf::length

the bunch length

Definition at line 267 of file bpm_interface.h.

Referenced by get_mode_amplitude().

7.6.2.6 double bunchconf::arrival_time

arrival time of bunch

Definition at line 268 of file bpm_interface.h.

Referenced by generate_bpmsignal().

7.6.2.7 double bunchconf::position[2]

the bunch position x,y at the bpm coo

Definition at line 269 of file bpm_interface.h.

Referenced by get_bpmhit().

7.6.2.8 double bunchconf::slope[2]

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

Definition at line 270 of file bpm_interface.h.

Referenced by get_bpmhit().

7.6.2.9 double bunchconf::tilt[2]

the bunch tilt x',y' at the bpm coo

Definition at line 271 of file bpm_interface.h.

7.6.2.10 double bunchconf::bpmposition[3]

where the beam hits the BPM in the BPM local co

Definition at line 273 of file bpm_interface.h.

Referenced by get_bpmhit(), get_mode_amplitude(), and setup_calibration().

7.6.2.11 double bunchconf::bpmslope[2]

slope of beam through the BPM in BPM local co

Definition at line 274 of file bpm_interface.h.

Referenced by get_bpmhit(), and get_mode_amplitude().

7.6.2.12 double bunchconf::bpmtilt[2]

bunch tilt in the BPM local co

Definition at line 275 of file bpm_interface.h.

Referenced by get_bpmhit().

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

- bpminterface/**bpm_interface.h**

7.7 complex_t Struct Reference

`#include <bpm_nr.h>`

7.7.1 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**

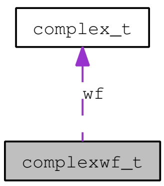
The documentation for this struct was generated from the following file:

- bpmnr/**bpm_nr.h**

7.8 complexwf_t Struct Reference

`#include <bpm_wf.h>`

Collaboration diagram for complexwf_t:



7.8.1 Detailed Description

Structure representing a waveform of complex numbers

Definition at line 188 of file bpm_wf.h.

Data Fields

- int **ns**
- double **fs**
- **complex_t** ∗ **wf**

7.8.2 Field Documentation

7.8.2.1 int complexwf_t::ns

The number of samples in the waveform

Definition at line 189 of file bpm_wf.h.

Referenced by add_mode_response(), complexfft(), complexwf(), complexwf_add(), complexwf_add_ampnoise(), complexwf_add_cwtone(), complexwf_add_dcywave(), complexwf_add_noise(), complexwf_add_phasenoise(), complexwf_bias(), complexwf_compat(), complexwf_copy(), complexwf_copy_new(), complexwf_divide(), complexwf_getamp(), complexwf_getamp_new(), complexwf_getimag(), complexwf_getimag_new(), complexwf_getphase(), complexwf_getphase_new(), complexwf_getreal(), complexwf_getreal_new(), complexwf_multiply(), complexwf_print(), complexwf_reset(), complexwf_scale(), complexwf_setfunction(), complexwf_setimag(), complexwf_setreal(), complexwf_setvalues(), complexwf_subset(), complexwf_subtract(), ddc(), fit_fft(), fit_fft_prepare(), generate_bpmsignal(), get_mode_response(), and realfft().

7.8.2.2 double complexwf_t::fs

The sampling frequency

Definition at line 190 of file bpm_wf.h.

Referenced by complexwf(), complexwf_add_cwtone(), complexwf_add_dcywave(), complexwf_compat(), complexwf_copy_new(), complexwf_getamp_new(), complexwf_getimag_new(), complexwf_getphase_new(), complexwf_getreal_new(), complexwf_print(), complexwf_setfunction(), complexwf_subset(), ddc(), fit_fft(), fit_fft_prepare(), generate_bpmsignal(), and get_mode_response().

7.8.2.3 complex_t* complexwf_t::wf

Pointer to an array of integers which hold the samples

Definition at line 191 of file bpm_wf.h.

Referenced by add_mode_response(), complexfft(), complexwf(), complexwf_add(), complexwf_add_ampnoise(), complexwf_add_cwtone(), complexwf_add_dcywave(), complexwf_add_noise(), complexwf_add_phasenoise(), complexwf_bias(), complexwf_copy(), complexwf_copy_new(), complexwf_delete(), complexwf_divide(), complexwf_getamp(), complexwf_getamp_new(), complexwf_getimag(), complexwf_getimag_new(), complexwf_getphase(), complexwf_getphase_new(), complexwf_getreal(), complexwf_getreal_new(), complexwf_multiply(), complexwf_print(), complexwf_reset(), complexwf_scale(), complexwf_setfunction(), complexwf_setimag(), complexwf_setreal(), complexwf_setvalues(), complexwf_subset(), complexwf_subtract(), downmix_waveform(), fit_fft(), fit_fft_prepare(), process_caltone(), process_waveform(), and realfft().

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

- bpmwf/**bpm_wf.h**

7.9 doublewf_t Struct Reference

```
#include <bpm_wf.h>
```

7.9.1 Detailed Description

Structure representing a waveform of doubles

Definition at line 174 of file bpm_wf.h.

Data Fields

- int **ns**
- double **fs**
- double * **wf**

7.9.2 Field Documentation

7.9.2.1 int doublewf_t::ns

The number of samples in the waveform

Definition at line 175 of file bpm_wf.h.

Referenced by add_mode_response(), check_saturation(), complexwf_getamp(), complexwf_getimag(), complexwf_getphase(), complexwf_getreal(), complexwf_setimag(), complexwf_setreal(), ddc(), digitise(), doublewf(), doublewf_add(), doublewf_add_ampnoise(), doublewf_add_cwtone(), doublewf_add_dcywave(), doublewf_basic_stats(), doublewf_bias(), doublewf_compat(), doublewf_copy(), doublewf_copy_new(), doublewf_derive(), doublewf_divide(), doublewf_frequency_series(), doublewf_getvalue(), doublewf_integrate(), doublewf_multiply(), doublewf_print(), doublewf_resample(), doublewf_reset(), doublewf_sample_series(), doublewf_scale(), doublewf_setvalues(), doublewf_subset(), doublewf_subtract(), doublewf_time_series(), downmix_waveform(), fit_waveform(), generate_bpmsignal(), generate_diodesignal(), get_t0(), intwf_cast_new(), process_diode(), and rf_rectify().

7.9.2.2 double doublewf_t::fs

The sampling frequency

Definition at line 176 of file bpm_wf.h.

Referenced by ddc(), digitise(), doublewf(), doublewf_add_cwtone(), doublewf_add_dcywave(), doublewf_compat(), doublewf_copy_new(), doublewf_derive(), doublewf_frequency_series(), doublewf_getvalue(), doublewf_integrate(), doublewf_print(), doublewf_resample(), doublewf_subset(), doublewf_time_series(), downmix_waveform(), fit_waveform(), generate_bpmsignal(), generate_diodesignal(), get_t0(), intwf_cast_new(), and process_diode().

7.9.2.3 double* doublewf_t::wf

Pointer to an array of doubles which hold the samples

Definition at line 177 of file bpm_wf.h.

Referenced by add_mode_response(), apply_filter(), check_saturation(), complexwf_getamp(), complexwf_getamp_new(), complexwf_getimag(), complexwf_getimag_new(), complexwf_getphase(), complexwf_getphase_new(), complexwf_getreal(), complexwf_getreal_new(), complexwf_setimag(), complexwf_setreal(), ddc(), doublewf(), doublewf_add(), doublewf_add_ampnoise(), doublewf_add_cwtone(), doublewf_add_dcywave(), doublewf_basic_stats(), doublewf_bias(), doublewf_cast(), doublewf_cast_new(), doublewf_copy(), doublewf_copy_new(), doublewf_delete(), doublewf_derive(), doublewf_divide(), doublewf_frequency_series(), doublewf_getvalue(), doublewf_integrate(), doublewf_multiply(), doublewf_print(), doublewf_resample(), doublewf_reset(), doublewf_sample_series(), doublewf_scale(), doublewf_setvalues(), doublewf_subset(), doublewf_subtract(), doublewf_time_series(), downmix_waveform(), filter_impulse_response(), filter_step_response(), fit_waveform(), generate_bpmsignal(), generate_diodesignal(), get_mode_response(), get_t0(), intwf_cast(), intwf_cast_new(), process_diode(), realfft(), and rf_rectify().

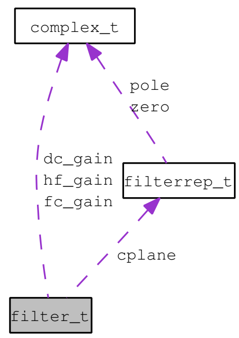
The documentation for this struct was generated from the following file:

- bpmwf/**bpm_wf.h**

7.10 filter_t Struct Reference

`#include <bpm_dsp.h>`

Collaboration diagram for filter_t:



7.10.1 Detailed Description

The filter structure.

Definition at line 437 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**
- double **gauss_cutoff**
- **complex_t dc_gain**
- **complex_t fc_gain**
- **complex_t hf_gain**
- double **gain**
- **filterrep_t** ∗ **cplane**
- int **nxc**
- double **xc** [MAXPZ+1]

- int **nxc_ac**
- double **xc_ac** [MAXPZ+1]
- int **nyc**
- double **yc** [MAXPZ+1]
- int **nyc_ac**
- double **yc_ac** [MAXPZ+1]
- double **xv** [MAXPZ+1]
- double **xv_ac** [MAXPZ+1]
- double **yv** [MAXPZ+1]
- double **yv_ac** [MAXPZ+1]
- int **ns**
- double ∗ **wfbuffer**

7.10.2 Field Documentation

7.10.2.1 char filter_t::name[80]

The filter's name

Definition at line 438 of file bpm_dsp.h.

Referenced by create_filter(), and print_filter().

7.10.2.2 unsigned int filter_t::options

type and option bits for filter

Definition at line 440 of file bpm_dsp.h.

Referenced by apply_filter(), calculate_filter_coefficients(), create_filter(), create_resonator_-representation(), create_splane_representation(), gaussian_filter_coeffs(), normalise_filter(), print_filter(), and zplane_transform().

7.10.2.3 int filter_t::order

filter order

Definition at line 441 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

7.10.2.4 double filter_t::fs

sampling frequency

Definition at line 443 of file bpm_dsp.h.

Referenced by create_filter(), and gaussian_filter_coeffs().

7.10.2.5 double filter_t::f1

first frequency (left edge for bandpass/stop)

Definition at line 444 of file bpm_dsp.h.

Referenced by create_filter(), and gaussian_filter_coeffs().

7.10.2.6 double filter_t::f2

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

Definition at line 445 of file bpm_dsp.h.

Referenced by create_filter().

7.10.2.7 double filter_t::alpha1

rescaled f1

Definition at line 447 of file bpm_dsp.h.

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

7.10.2.8 double filter_t::alpha2

rescaled f2

Definition at line 448 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and create_filter().

7.10.2.9 double filter_t::w_alpha1

warped alpha1

Definition at line 450 of file bpm_dsp.h.

Referenced by create_filter(), and normalise_filter().

7.10.2.10 double filter_t::w_alpha2

warped alpha2

Definition at line 451 of file bpm_dsp.h.

Referenced by create_filter(), and normalise_filter().

7.10.2.11 double filter_t::cheb_ripple

ripple for chebyshev filters

Definition at line 453 of file bpm_dsp.h.

Referenced by create_filter(), and create_splane_representation().

7.10.2.12 double filter_t::Q

Q factor for resonators

Definition at line 454 of file bpm_dsp.h.

Referenced by create_filter(), and create_resonator_representation().

7.10.2.13 double filter_t::gauss_cutoff

gaussian filter cutoff parameter

Definition at line 455 of file bpm_dsp.h.

Referenced by create_filter(), and gaussian_filter_coeffs().

7.10.2.14 complex_t filter_t::dc_gain

Complex DC gain of the filter

Definition at line 457 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and print_filter().

7.10.2.15 complex_t filter_t::fc_gain

Complex Center frequency gain of filter

Definition at line 458 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and print_filter().

7.10.2.16 complex_t filter_t::hf_gain

Complex High frequency (fNy) gain of filter

Definition at line 459 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), and print_filter().

7.10.2.17 double filter_t::gain

Actual Filter gain

Definition at line 460 of file bpm_dsp.h.

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

7.10.2.18 filterrep_t* filter_t::cplane

pointer to complex filter representation, poles and zeros

Definition at line 462 of file bpm_dsp.h.

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

7.10.2.19 int filter_t::nxc

number of x coefficients

Definition at line 464 of file bpm_dsp.h.

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

7.10.2.20 double filter_t::xc[MAXPZ+1]

pointer to array of x coefficients

Definition at line 465 of file bpm_dsp.h.

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

7.10.2.21 int filter_t::nxc_ac

number of anti-causal x coefficients

Definition at line 467 of file bpm_dsp.h.

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

7.10.2.22 double filter_t::xc_ac[MAXPZ+1]

pointer to array of anti-causal x coefficients

Definition at line 468 of file bpm_dsp.h.

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

7.10.2.23 int filter_t::nyc

number of y coefficients (for IIR filters)

Definition at line 470 of file bpm_dsp.h.

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

7.10.2.24 double filter_t::yc[MAXPZ+1]

pointer to array of y coefficients

Definition at line 471 of file bpm_dsp.h.

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

7.10.2.25 int filter_t::nyc_ac

number of anti-causal y coefficients (for IIR filters)

Definition at line 473 of file bpm_dsp.h.

7.10.2.26 double filter_t::yc_ac[MAXPZ+1]

pointer to array of anti-causal y coefficients

Definition at line 474 of file bpm_dsp.h.

7.10.2.27 double filter_t::xv[MAXPZ+1]

filter x buffer, used in apply_filter

Definition at line 476 of file bpm_dsp.h.

Referenced by apply_filter().

7.10.2.28 double filter_t::xv_ac[MAXPZ+1]

filter x buffer, used in apply_filter

Definition at line 477 of file bpm_dsp.h.

Referenced by apply_filter().

7.10.2.29 double filter_t::yv[MAXPZ+1]

filter y buffer, used in apply_filter

Definition at line 479 of file bpm_dsp.h.

Referenced by apply_filter().

7.10.2.30 double filter_t::yv_ac[MAXPZ+1]

filter y buffer, used in apply_filter

Definition at line 480 of file bpm_dsp.h.

Referenced by apply_filter().

7.10.2.31 int filter_t::ns

number of samples of waveforms to be filtered

Definition at line 482 of file bpm_dsp.h.

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

7.10.2.32 double* filter_t::wfbuffer

waveform buffer for filter computations, allocated once !

Definition at line 483 of file bpm_dsp.h.

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

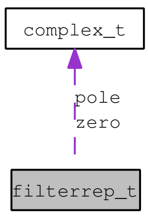
The documentation for this struct was generated from the following file:

- bpmdsp/**bpm_dsp.h**

7.11 filterrep_t Struct Reference

```
#include <bpm_dsp.h>
```

Collaboration diagram for filterrep_t:



7.11.1 Detailed Description

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

Definition at line 427 of file bpm_dsp.h.

Data Fields

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

7.11.2 Field Documentation

7.11.2.1 int filterrep_t::npoles

The number of filter poles

Definition at line 428 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), create_filter(), create_resonator_representation(), create_-splane_representation(), normalise_filter(), print_filter_representation(), and zplane_transform().

7.11.2.2 int filterrep_t::nzeros

The number of filter zeros

Definition at line 429 of file bpm_dsp.h.

Referenced by calculate_filter_coefficients(), create_resonator_representation(), normalise_filter(), print_-filter_representation(), and zplane_transform().

7.11.2.3 complex_t filterrep_t::pole[MAXPZ]

Array of the filter's complex poles

Definition at line 430 of file bpm_dsp.h.

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

7.11.2.4 complex_t filterrep_t::zero[MAXPZ]

Array of the filter's complex zeros

Definition at line 431 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**

7.12 intwf_t Struct Reference

```
#include <bpm_wf.h>
```

7.12.1 Detailed Description

Structure representing a waveform of integers

Definition at line 181 of file bpm_wf.h.

Data Fields

- int **ns**
- double **fs**
- int * **wf**

7.12.2 Field Documentation

7.12.2.1 int intwf_t::ns

The number of samples in the waveform

Definition at line 182 of file bpm_wf.h.

Referenced by digitise(), doublewf_cast(), doublewf_cast_new(), intwf(), intwf_add(), intwf_add_-ampnoise(), intwf_add_cwtone(), intwf_add_dcywave(), intwf_bias(), intwf_cast(), intwf_cast_new(), intwf_compat(), intwf_copy(), intwf_copy_new(), intwf_derive(), intwf_divide(), intwf_integrate(), intwf_multiply(), intwf_print(), intwf_resample(), intwf_reset(), intwf_sample_series(), intwf_scale(), intwf_setvalues(), intwf_subset(), and intwf_subtract().

7.12.2.2 double intwf_t::fs

The sampling frequency

Definition at line 183 of file bpm_wf.h.

Referenced by digitise(), doublewf_cast_new(), intwf(), intwf_add_cwtone(), intwf_add_dcywave(), intwf_compat(), intwf_copy_new(), intwf_derive(), intwf_integrate(), intwf_print(), intwf_resample(), and intwf_subset().

7.12.2.3 int* intwf_t::wf

Pointer to an array of integers which hold the samples

Definition at line 184 of file bpm_wf.h.

Referenced by digitise(), doublewf_cast(), doublewf_cast_new(), intwf(), intwf_add(), intwf_add_-ampnoise(), intwf_add_cwtone(), intwf_add_dcywave(), intwf_bias(), intwf_cast(), intwf_cast_new(), intwf_copy(), intwf_copy_new(), intwf_delete(), intwf_derive(), intwf_divide(), intwf_integrate(), intwf_-multiply(), intwf_print(), intwf_resample(), intwf_reset(), intwf_sample_series(), intwf_scale(), intwf_-setvalues(), intwf_subset(), and intwf_subtract().

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

- bpmwf/**bpm_wf.h**

7.13 lm_fstate Struct Reference

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

7.13.1 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**

7.14 m33 Struct Reference

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

7.14.1 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]

7.14.2 Field Documentation

7.14.2.1 double m33::e[3][3]

the matrix

Definition at line 50 of file bpm_orbit.h.

Referenced by generate_diodesignal(), m_matadd(), m_matmult(), m_print(), m_rotmat(), and v_-matmult().

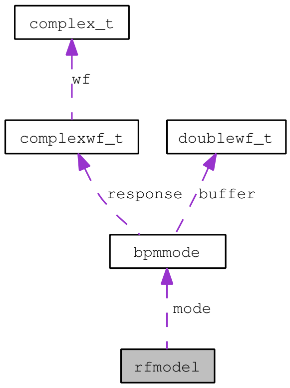
The documentation for this struct was generated from the following file:

- bpmorbit/**bpm_orbit.h**

7.15 rfmodel Struct Reference

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

Collaboration diagram for rfmodel:



7.15.1 Detailed Description

This structure contains the complete RF model for a BPM, which is essentially a collection of it's resonant modes and sensitivities

Definition at line 296 of file bpm_interface.h.

Data Fields

- char **name** [20]
- int **nmodes**
- **bpmmode_t** ∗ **mode**

7.15.2 Field Documentation

7.15.2.1 char rfmodel::name[20]

A name for the cavity's RF model

Definition at line 297 of file bpm_interface.h.

7.15.2.2 int rfmodel::nmodes

The number of BPM modes in the model

Definition at line 298 of file bpm_interface.h.

7.15.2.3 bpmmode_t∗ rfmodel::mode

A list of pointers to the array of modes

Definition at line 299 of file bpm_interface.h.

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

- bpminterface/**bpm_interface.h**

7.16 v3 Struct Reference

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

7.16.1 Detailed Description

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

Definition at line 39 of file bpm_orbit.h.

Data Fields

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

7.16.2 Field Documentation

7.16.2.1 double v3::x

x-coordinate

Definition at line 40 of file bpm_orbit.h.

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

7.16.2.2 double v3::y

y-coordinate

Definition at line 41 of file bpm_orbit.h.

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

7.16.2.3 double v3::z

z-coordinate

Definition at line 42 of file bpm_orbit.h.

Referenced by get_bpmhit(), 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**

7.17 wfstat_t Struct Reference

```
#include <bpm_wf.h>
```

7.17.1 Detailed Description

Structure with basic waveform statistics

Definition at line 196 of file bpm_wf.h.

Data Fields

- int **imax**
- int **imin**
- double **max**
- double **min**
- double **mean**
- double **rms**

7.17.2 Field Documentation

7.17.2.1 int wfstat_t::imax

The sample nr of maximum of waveform

Definition at line 197 of file bpm_wf.h.

Referenced by doublewf_basic_stats(), wfstat_print(), and wfstat_reset().

7.17.2.2 int wfstat_t::imin

The sample nr of minimum of waveform

Definition at line 198 of file bpm_wf.h.

Referenced by doublewf_basic_stats(), wfstat_print(), and wfstat_reset().

7.17.2.3 double wfstat_t::max

The maximum value of waveform

Definition at line 199 of file bpm_wf.h.

Referenced by doublewf_basic_stats(), wfstat_print(), and wfstat_reset().

7.17.2.4 double wfstat_t::min

The minimum value of waveform

Definition at line 200 of file bpm_wf.h.

Referenced by doublewf_basic_stats(), wfstat_print(), and wfstat_reset().

7.17.2.5 double wfstat_t::mean

The mean of waveform

Definition at line 201 of file bpm_wf.h.

Referenced by doublewf_basic_stats(), get_pedestal(), process_diode(), wfstat_print(), and wfstat_reset().

7.17.2.6 double wfstat_t::rms

The rms of waveform

Definition at line 202 of file bpm_wf.h.

Referenced by doublewf_basic_stats(), get_pedestal(), process_diode(), wfstat_print(), and wfstat_reset().

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

- bpmwf/**bpm_wf.h**

8 File Documentation

8.1 bpm_units.h File Reference

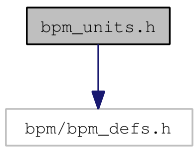
8.1.1 Detailed Description

Physical unit definitions for libbpm.

Definition in file **bpm_units.h**.

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

Include dependency graph for bpm_units.h:



Defines

- #define _**cent**__
- #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 _**degrees**__
- #define _**mC**__

- #define **_uC__**
- #define **_nC__**
- #define **_pC__**
- #define **_meter__**
- #define **_mmeter__**
- #define **_umeter__**
- #define **_nmeter__**
- #define **_Volt__**
- #define **_mVolt__**
- #define **_uVolt__**
- #define **_nVolt__**
- #define **_cLight__**

8.2 bpmanalysis/ana_compute_residual.c File Reference

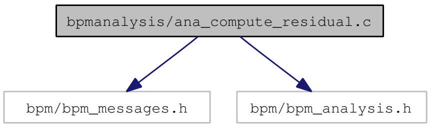
8.2.1 Detailed Description

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)

8.3 bpmanalysis/ana_def_cutfn.c File Reference

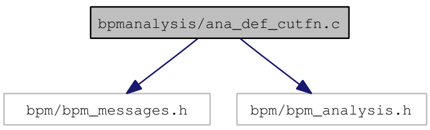
8.3.1 Detailed Description

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)

8.4 bpmanalysis/ana_get_svd_coeffs.c File Reference

8.4.1 Detailed Description

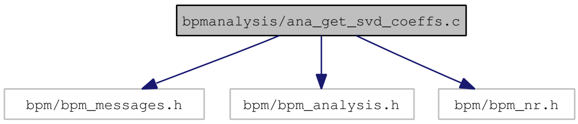
Definition in file **ana_get_svd_coeffs.c**.

`#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)

8.5 bpmanalysis/ana_set_cutfn.c File Reference

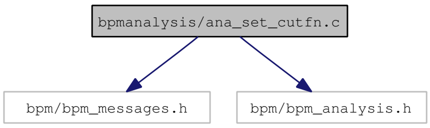
8.5.1 Detailed Description

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))

8.6 bpmanalysis/bpm_analysis.h File Reference

8.6.1 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...

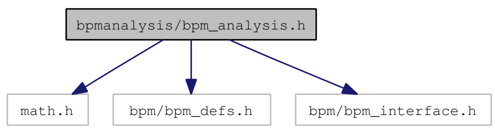
Definition in file **bpm_analysis.h**.

`#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)

8.7 bpmcalibration/bpm_calibration.h File Reference

8.7.1 Detailed Description

calibration routines

This header contains some BPM calibration routines

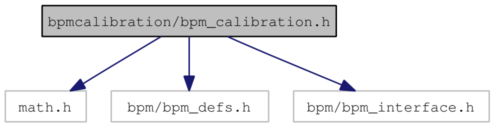
Definition in file **bpm_calibration.h**.

#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, **bunchconf_t** ∗bunch)
- EXTERN int **calibrate** (**bpmconf_t** ∗bpm, **bunchconf_t** ∗bunch, **bpmproc_t** ∗proc, int npulses, **bpmcalib_t** ∗cal)

8.8 bpmcalibration/calibrate.c File Reference

8.8.1 Detailed Description

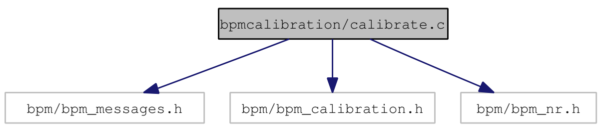
Definition in file **calibrate.c**.

#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, **bunchconf_t** ∗bunch, **bpmproc_t** ∗proc, int npulses, **bpmcalib_t** ∗cal)

8.9 bpmcalibration/setup_calibration.c File Reference

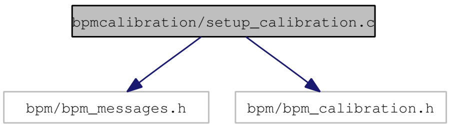
8.9.1 Detailed Description

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, **bunchconf_t** ∗bunch)

8.10 bpmdsp/bpm_dsp.h File Reference

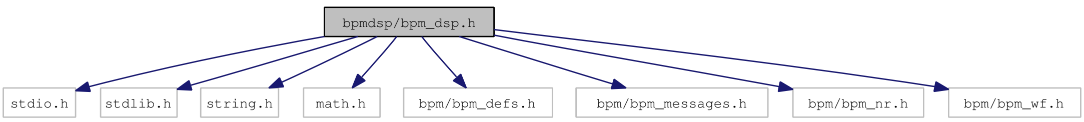
8.10.1 Detailed Description

libbpm digital signal processing routines

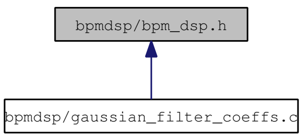
Definition in file **bpm_dsp.h**.

```
#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_nr.h"
#include "bpm/bpm_wf.h"
```

Include dependency graph for bpm_dsp.h:



This graph shows which files directly or indirectly include this file:



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 **CAUSAL**
- #define **ANTICAUSAL**
- #define **NONCAUSAL**
- #define **GAUSSIAN_SIGMA_BW**
- #define **LOWPASS**
- #define **HIGHPASS**
- #define **BANDPASS**
- #define **BANDSTOP**
- #define **NOTCH**
- #define **ALLPASS**
- #define **FIR**
- #define **IIR**
- #define **MAXORDER**
- #define **MAXPZ**
- #define **FILT_EPS**
- #define **MAX_RESONATOR_ITER**
- #define **FFT_FORWARD**
- #define **FFT_BACKWARD**

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, **doublewf_t** ∗w)
- EXTERN void **print_filter** (FILE ∗of, **filter_t** ∗f)
- EXTERN void **delete_filter** (**filter_t** ∗f)
- EXTERN int **filter_step_response** (**filter_t** ∗f, **doublewf_t** ∗w, int itrig)
- EXTERN int **filter_impulse_response** (**filter_t** ∗f, **doublewf_t** ∗w, 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 **gaussian_filter_coeffs** (**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)
- EXTERN int **ddc_initialise** (int ns, double fs)
- EXTERN void **ddc_cleanup** (void)

- int **ddc** (**doublewf_t** ∗w, double f, **filter_t** ∗filter, **complexwf_t** ∗dcw, **doublewf_t** ∗bufre, **doublewf_t** ∗bufim)
- EXTERN int **fft_gen_tables** (void)
- EXTERN int **fft_initialise** (int ns)
- EXTERN void **fft_cleanup** (void)
- EXTERN int **complexfft** (**complexwf_t** ∗z, int fft_mode)
- EXTERN int **realfft** (**doublewf_t** ∗y, int fft_mode, **complexwf_t** ∗z)
- EXTERN void **norm_phase** (double ∗phase)

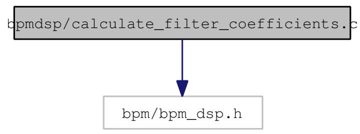
8.11 bpmdsp/calculate_filter_coefficients.c File Reference

8.11.1 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)

8.12 bpmdsp/create_filter.c File Reference

8.12.1 Detailed Description

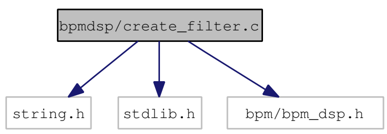
Definition in file **create_filter.c**.

#include <string.h>

#include <stdlib.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)

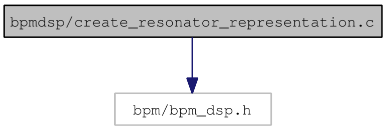
8.13 bpmdsp/create_resonator_representation.c File Reference

8.13.1 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)

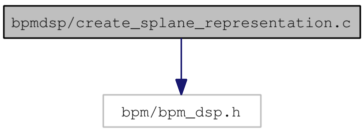
8.14 bpmdsp/create_splane_representation.c File Reference

8.14.1 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)

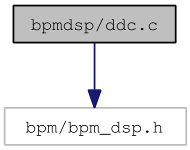
8.15 bpmdsp/ddc.c File Reference

8.15.1 Detailed Description

Definition in file **ddc.c**.

`#include "bpm/bpm_dsp.h"`

Include dependency graph for ddc.c:



Functions

- int **_check_ddc_buffers** (int ns, double fs)
- int **ddc_initialise** (int ns, double fs)
- void **ddc_cleanup** (void)
- int **ddc** (**doublewf_t** ∗w, double f, **filter_t** ∗filter, **complexwf_t** ∗dcw, **doublewf_t** ∗bufre, **doublewf_t** ∗bufim)

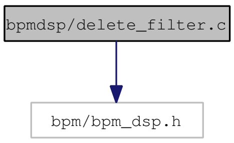
8.16 bpmdsp/delete_filter.c File Reference

8.16.1 Detailed Description

Definition in file **delete_filter.c**.

`#include "bpm/bpm_dsp.h"`

Include dependency graph for delete_filter.c:



Functions

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

8.17 bpmdsp/discrete_fourier_transforms.c File Reference

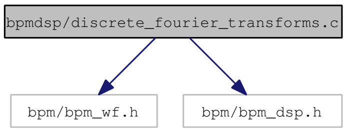
8.17.1 Detailed Description

Definition in file **discrete_fourier_transforms.c**.

`#include "bpm/bpm_wf.h"`

`#include "bpm/bpm_dsp.h"`

Include dependency graph for discrete_fourier_transforms.c:



Functions

- void **cdft** (int, int, double ∗, int ∗, double ∗)
- void **rdft** (int, int, double ∗, int ∗, double ∗)
- int **_is_pow2** (int n)
- int **_check_fft_buffers** (int ns)
- int **fft_gen_tables** (void)
- int **fft_initialise** (int ns)
- void **fft_cleanup** (void)
- int **complexfft** (**complexwf_t** ∗z, int fft_mode)
- int **realfft** (**doublewf_t** ∗y, int fft_mode, **complexwf_t** ∗z)

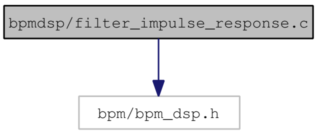
8.18 bpmdsp/filter_impulse_response.c File Reference

8.18.1 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, **doublewf_t** ∗w, int itrig)

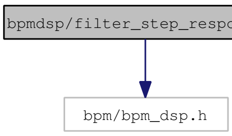
8.19 bpmdsp/filter_step_response.c File Reference

8.19.1 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, **doublewf_t** ∗w, int itrig)

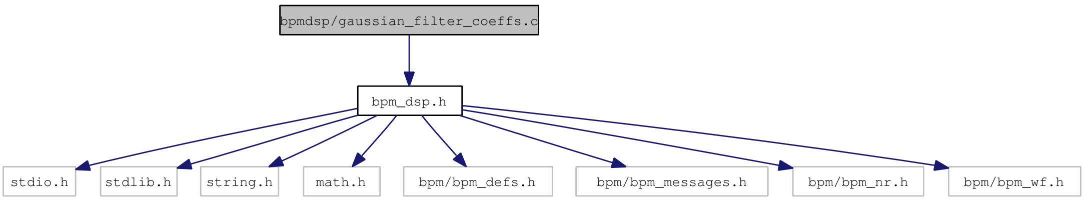
8.20 bpmdsp/gaussian_filter_coeffs.c File Reference

8.20.1 Detailed Description

Definition in file **gaussian_filter_coeffs.c**.

`#include "bpm_dsp.h"`

Include dependency graph for gaussian_filter_coeffs.c:



Functions

- int **gaussian_filter_coeffs** (**filter_t** ∗f)

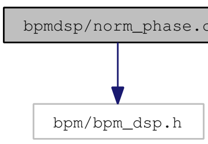
8.21 bpmdsp/norm_phase.c File Reference

8.21.1 Detailed Description

Definition in file **norm_phase.c**.

`#include <bpm/bpm_dsp.h>`

Include dependency graph for norm_phase.c:



Functions

- void **norm_phase** (double ∗phase)

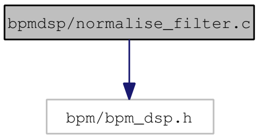
8.22 bpmdsp/normalise_filter.c File Reference

8.22.1 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)

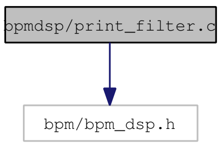
8.23 bpmdsp/print_filter.c File Reference

8.23.1 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)

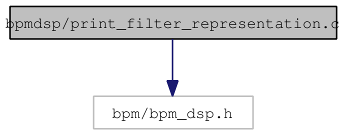
8.24 bpmdsp/print_filter_representation.c File Reference

8.24.1 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)

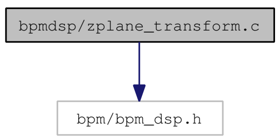
8.25 bpmdsp/zplane_transform.c File Reference

8.25.1 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)

8.26 bpminterface/bpm_interface.h File Reference

8.26.1 Detailed Description

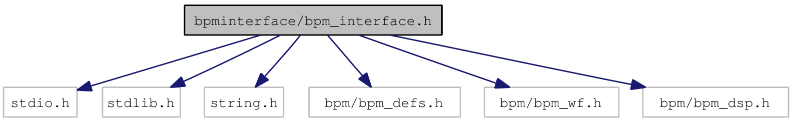
Front end interface structure definitions and handlers.

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 <bpm/bpm_wf.h>
#include <bpm/bpm_dsp.h>
```

Include dependency graph for bpm_interface.h:



Data Structures

- struct **bpmconf**
- struct **bpmcalib**
- struct **bpmproc**
- struct **beamconf**
- struct **bunchconf**
- struct **bpmmode**
- struct **rfmodel**

Typedefs

- typedef struct **bpmconf bpmconf_t**
- typedef struct **bpmcalib bpmcalib_t**
- typedef struct **bpmproc bpmproc_t**
- typedef struct **beamconf beamconf_t**
- typedef struct **bunchconf bunchconf_t**
- typedef struct **bpmmode bpmmode_t**
- typedef struct **rfmodel rfmodel_t**
- typedef enum **triggertype triggertype_t**

Enumerations

- enum **bpmtype_t** { **diode**, **monopole**, **dipole** }
- enum **triggertype** { **positive**, **negative**, **bipolar** }
- enum **bpmpol_t** { **horiz**, **vert** }
- enum **bpmphase_t** { **randomised**, **locked** }

Variables

- EXTERN int **bpm_verbose**
- EXTERN int **libbpm_evtnum**

8.27 bpmmessages/bpm_error.c File Reference

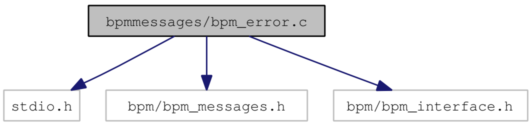
8.27.1 Detailed Description

Definition in file **bpm_error.c**.

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

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

Include dependency graph for bpm_error.c:



Functions

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

8.28 bpmmessages/bpm_messages.h File Reference

8.28.1 Detailed Description

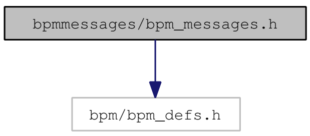
libbpm error/warning messages

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)

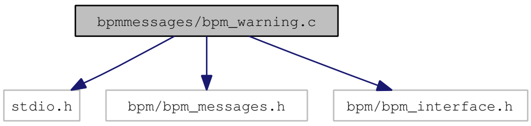
8.29 bpmmessages/bpm_warning.c File Reference

8.29.1 Detailed Description

Definition in file **bpm_warning.c**.

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

Include dependency graph for bpm_warning.c:



Functions

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

8.30 bpmnr/bpm_nr.h File Reference

8.30.1 Detailed Description

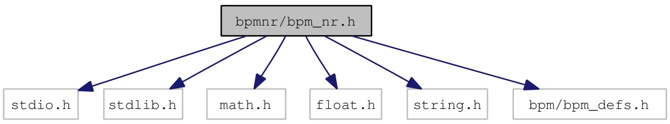
libbpm numerical helper routines

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

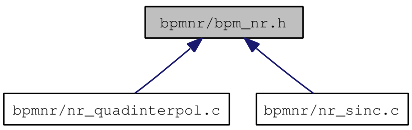
Definition in file **bpm_nr.h**.

```
#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:



This graph shows which files directly or indirectly include this file:



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 struct 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_abs2** (**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)
- EXTERN double **nr_quadinterpol** (double x, double x1, double x2, double x3, double y1, double y2, double y3)
- EXTERN double **sinc** (double x)
- EXTERN double **lanczos** (double x, int a)
- EXTERN double **dround** (double x)

Variables

- EXTERN long **bpm_rseed**

8.31 bpmnr/dround.c File Reference

8.31.1 Detailed Description

Definition in file **dround.c**.

Functions

- double **dround** (double x)

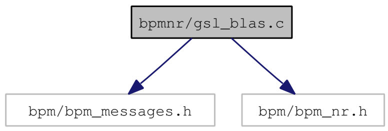
8.32 bpmnr/gsl_blas.c File Reference

8.32.1 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)

8.33 bpmnr/gsl_block.c File Reference

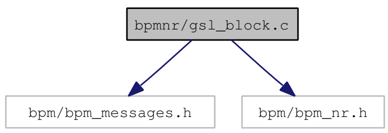
8.33.1 Detailed Description

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)

8.34 bpmnr/gsl_eigen.c File Reference

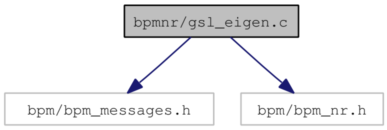
8.34.1 Detailed Description

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)

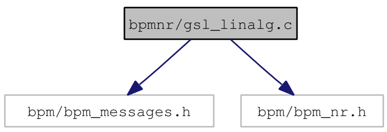
8.35 bpmnr/gsl_linalg.c File Reference

8.35.1 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)

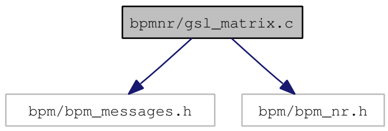
8.36 bpmnr/gsl_matrix.c File Reference

8.36.1 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)

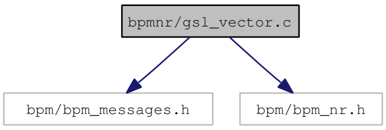
8.37 bpmnr/gsl_vector.c File Reference

8.37.1 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)

8.38 bpmnr/nr_checks.c File Reference

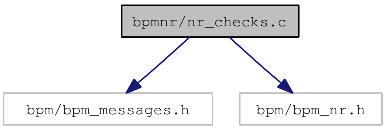
8.38.1 Detailed Description

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)

8.38.2 Function Documentation

8.38.2.1 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().

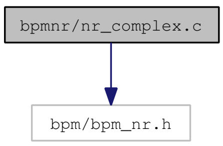
8.39 bpmnr/nr_complex.c File Reference

8.39.1 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_abs2** (**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)

8.40 bpmnr/nr_fit.c File Reference

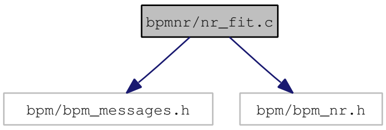
8.40.1 Detailed Description

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)

8.41 bpmnr/nr_four1.c File Reference

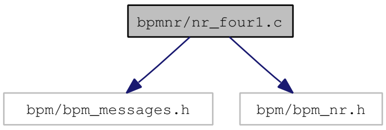
8.41.1 Detailed Description

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)

8.42 bpmnr/nr_gammln.c File Reference

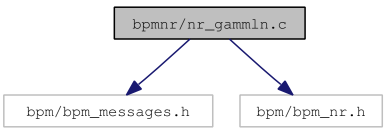
8.42.1 Detailed Description

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)

8.43 bpmnr/nr_gammq.c File Reference

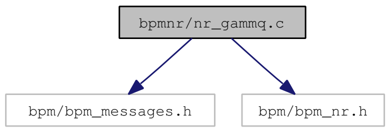
8.43.1 Detailed Description

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)

8.44 bpmnr/nr_gcf.c File Reference

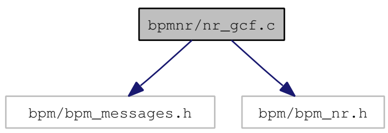
8.44.1 Detailed Description

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)

8.45 bpmnr/nr_gser.c File Reference

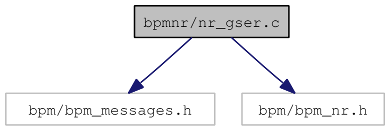
8.45.1 Detailed Description

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)

8.46 bpmnr/nr_levmar.c File Reference

8.46.1 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.

//

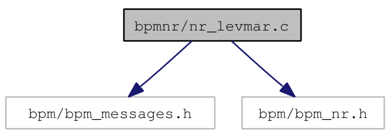
Changes: BM. Modified the names of the routines somewhat to have them correspond to the rest of libbpm

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)

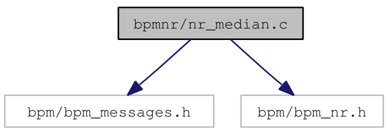
8.47 bpmnr/nr_median.c File Reference

8.47.1 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)

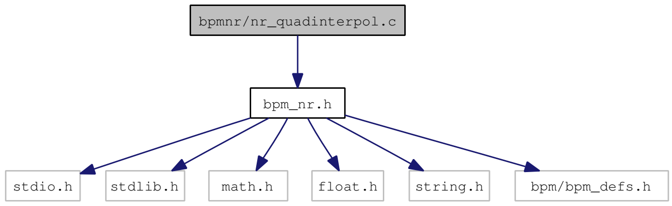
8.48 bpmnr/nr_quadinterpol.c File Reference

8.48.1 Detailed Description

Definition in file **nr_quadinterpol.c**.

`#include "bpm_nr.h"`

Include dependency graph for nr_quadinterpol.c:



Functions

- double **nr_quadinterpol** (double x, double x1, double x2, double x3, double y1, double y2, double y3)

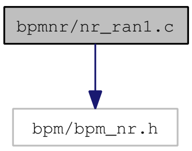
8.49 bpmnr/nr_ran1.c File Reference

8.49.1 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)

8.50 bpmnr/nr_rangauss.c File Reference

8.50.1 Detailed Description

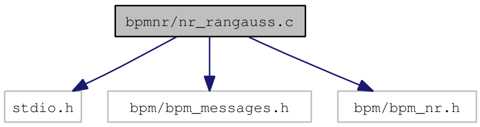
Definition in file **nr_rangauss.c**.

`#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)

8.51 bpmnr/nr_ranuniform.c File Reference

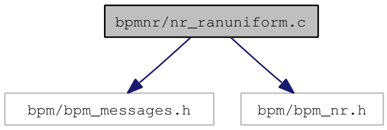
8.51.1 Detailed Description

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)

8.52 bpmnr/nr_realft.c File Reference

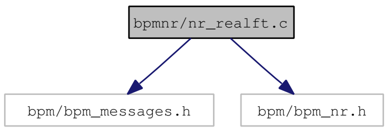
8.52.1 Detailed Description

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)

8.53 bpmnr/nr_seed.c File Reference

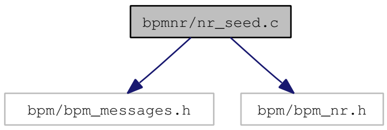
8.53.1 Detailed Description

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**

8.53.2 Variable Documentation

8.53.2.1 long bpm_rseed

the global random seed variable

Definition at line 9 of file nr_seed.c.

Referenced by nr_rangauss(), nr_ranuniform(), and nr_seed().

8.54 bpmnr/nr_select.c File Reference

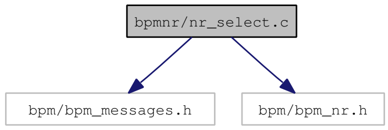
8.54.1 Detailed Description

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)

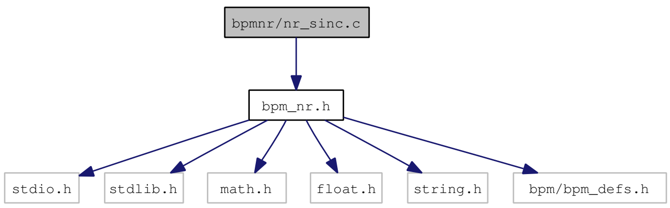
8.55 bpmnr/nr_sinc.c File Reference

8.55.1 Detailed Description

Definition in file **nr_sinc.c**.

```
#include "bpm_nr.h"
```

Include dependency graph for nr_sinc.c:



Functions

- double **sinc** (double x)
- double **lanczos** (double x, int a)

8.56 bpmorbit/bpm_orbit.h File Reference

8.56.1 Detailed Description

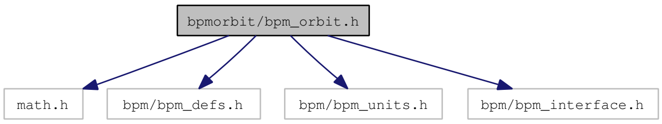
libbpm orbit generation routines

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

Include dependency graph for bpm_orbit.h:



Data Structures

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

Functions

- EXTERN double **get_rbend** (double e, double B, double l, double p)
- EXTERN double **get_sbend** (double e, double B, double l, double p)
- EXTERN int **get_bpmhit** (**bunchconf_t** ∗bunch, **bpmconf_t** ∗bpm)
- EXTERN int **get_bpmhits** (**beamconf_t** ∗beam, **bpmconf_t** ∗bpm)
- 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)

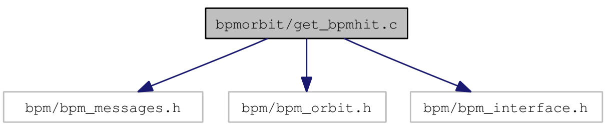
8.57 bpmorbit/get_bpmhit.c File Reference

8.57.1 Detailed Description

Definition in file **get_bpmhit.c**.

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

Include dependency graph for get_bpmhit.c:



Functions

- int **get_bpmhits** (**beamconf_t** ∗beam, **bpmconf_t** ∗bpm)
- int **get_bpmhit** (**bunchconf_t** ∗bunch, **bpmconf_t** ∗bpm)

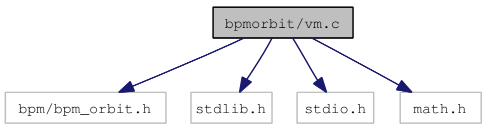
8.58 bpmorbit/vm.c File Reference

8.58.1 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)

8.59 bpmprocess/bpm_process.h File Reference

8.59.1 Detailed Description

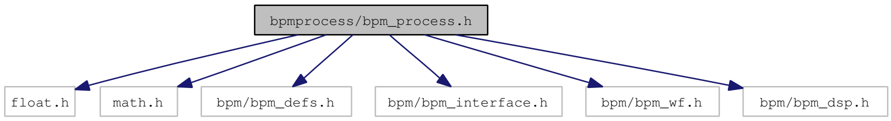
libbpm main processing routines

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 <bpm/bpm_wf.h>
#include <bpm/bpm_dsp.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_FULL**
- #define **PROC_FIT_DDC**
- #define **PROC_FIT_FFT**
- #define **PROC_RAW_PHASE**
- #define **PROC_CORR_AMP**
- #define **PROC_CORR_PHASE**
- #define **PROC_CORR_GAIN**

Functions

- EXTERN int **process_diode** (**doublewf_t** ∗signal, **bpmconf_t** ∗conf, **bpmproc_t** ∗proc)
- EXTERN int **process_monopole** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmcalib_t** ∗cal, **bpmproc_t** ∗proc, **bpmproc_t** ∗trig, unsigned int mode)
- EXTERN int **process_dipole** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmcalib_t** ∗cal, **bpmproc_t** ∗proc, **bpmproc_t** ∗trig, **bpmproc_t** ∗ampref, **bpmproc_t** ∗phaseref, unsigned int mode)
- EXTERN int **process_waveform** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmproc_t** ∗proc, **bpmproc_t** ∗trig, unsigned int mode)
- EXTERN int **postprocess_waveform** (**bpmconf_t** ∗bpm, **bpmproc_t** ∗proc, **bpmcalib_t** ∗cal, **bpmproc_t** ∗ampref, **bpmproc_t** ∗phaseref, unsigned int mode)
- EXTERN int **process_caltone** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmproc_t** ∗proc, unsigned int mode)
- EXTERN int **correct_gain** (**bpmproc_t** ∗proc, **bpmcalib_t** ∗cal, unsigned int mode)

- EXTERN int **fit_waveform** (**doublewf_t** ∗w, double t0, double i_freq, double i_tdecay, double i_amp, double i_phase, double ∗freq, double ∗tdecay, double ∗amp, double ∗phase)
- EXTERN int **fit_diodepulse** (**doublewf_t** ∗w, double ∗t0)
- EXTERN int **fft_waveform** (**doublewf_t** ∗w, **complexwf_t** ∗ft)
- EXTERN int **fit_fft_prepare** (**complexwf_t** ∗ft, int ∗n1, int ∗n2, double ∗amp, double ∗freq, double ∗fwhm)
- EXTERN int **fit_fft** (**complexwf_t** ∗ft, double ∗freq, double ∗tdecay, double ∗A, double ∗C)
- EXTERN int **check_saturation** (**doublewf_t** ∗w, int nbits, int ∗iunsat)
- EXTERN int **downmix_waveform** (**doublewf_t** ∗w, double frequency, **complexwf_t** ∗out)
- EXTERN int **ddc_waveform** (**doublewf_t** ∗w, double frequency, **filter_t** ∗filt, **complexwf_t** ∗dc, **doublewf_t** ∗buf_re, **doublewf_t** ∗buf_im)
- EXTERN int **ddc_sample_waveform** (**doublewf_t** ∗w, double frequency, **filter_t** ∗filt, int iSample, double t0, double tdecay, double ∗amp, double ∗phase, **doublewf_t** ∗buf_re, **doublewf_t** ∗buf_im)
- EXTERN int **get_pedestal** (**doublewf_t** ∗wf, int range, double ∗offset, double ∗rms)
- EXTERN int **get_t0** (**doublewf_t** ∗w, 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)

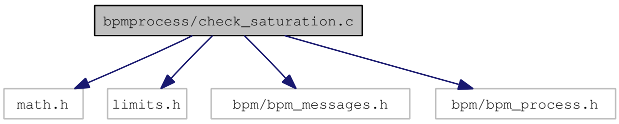
8.60 bpmprocess/check_saturation.c File Reference

8.60.1 Detailed Description

Definition in file **check_saturation.c**.

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

Include dependency graph for check_saturation.c:



Functions

- int **check_saturation** (**doublewf_t** ∗w, int nbits, int ∗iunsat)

8.61 bpmprocess/correct_gain.c File Reference

8.61.1 Detailed Description

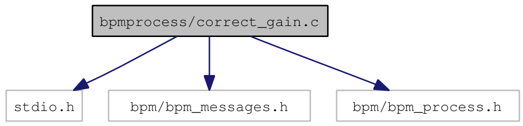
Definition in file **correct_gain.c**.

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

#include <bpm/bpm_messages.h>

#include <bpm/bpm_process.h>

Include dependency graph for correct_gain.c:



Functions

- int **correct_gain** (**bpmproc_t** ∗proc, **bpmcalib_t** ∗cal, unsigned int mode)

8.62 bpmprocess/ddc_sample_waveform.c File Reference

8.62.1 Detailed Description

Definition in file **ddc_sample_waveform.c**.

#include <stdio.h>

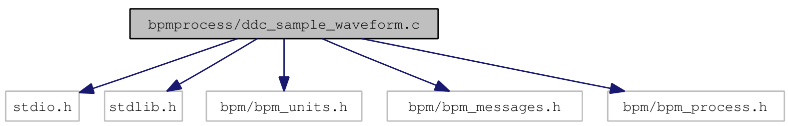
#include <stdlib.h>

#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** (**doublewf_t** ∗w, double frequency, **filter_t** ∗filt, int iSample, double t0, double tdecay, double ∗amp, double ∗phase, **doublewf_t** ∗buf_re, **doublewf_t** ∗buf_im)

8.63 bpmprocess/ddc_waveform.c File Reference

8.63.1 Detailed Description

Definition in file **ddc_waveform.c**.

#include <stdio.h>

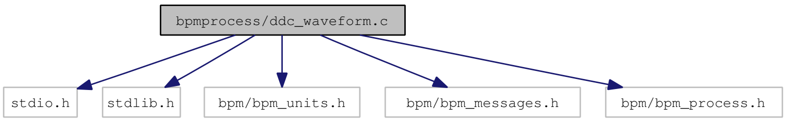
#include <stdlib.h>

#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** (**doublewf_t** ∗w, double frequency, **filter_t** ∗filt, **complexwf_t** ∗dc, **doublewf_t** ∗buf_re, **doublewf_t** ∗buf_im)

8.64 bpmprocess/downmix_waveform.c File Reference

8.64.1 Detailed Description

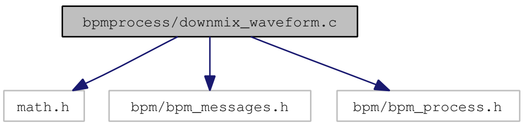
Definition in file **downmix_waveform.c**.

`#include <math.h>`

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_process.h>`

Include dependency graph for downmix_waveform.c:



Functions

- int **downmix_waveform** (**doublewf_t** ∗w, double freq, **complexwf_t** ∗out)

8.65 bpmprocess/fft_waveform.c File Reference

8.65.1 Detailed Description

Definition in file **fft_waveform.c**.

`#include <stdio.h>`

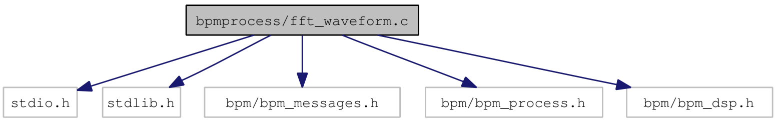
`#include <stdlib.h>`

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_process.h>`

`#include <bpm/bpm_dsp.h>`

Include dependency graph for fft_waveform.c:



Functions

- int **fft_waveform** (**doublewf_t** ∗w, **complexwf_t** ∗fft)

8.66 bpmprocess/fit_diodepulse.c File Reference

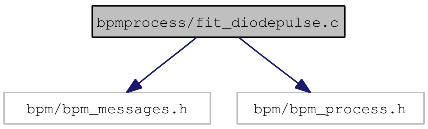
8.66.1 Detailed Description

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** (**doublewf_t** ∗w, double ∗t0)

8.67 bpmprocess/fit_fft.c File Reference

8.67.1 Detailed Description

Definition in file **fit_fft.c**.

`#include <stdio.h>`

`#include <stdlib.h>`

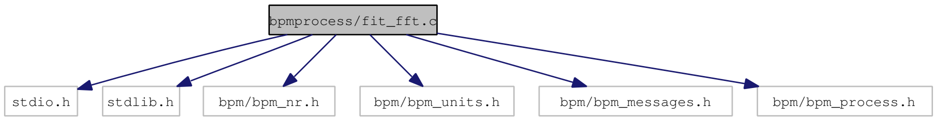
`#include <bpm/bpm_nr.h>`

`#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** (**complexwf_t** ∗ft, int ∗n1, int ∗n2, double ∗amp, double ∗freq, double ∗fwhm)
- int **fit_fft** (**complexwf_t** ∗ft, double ∗freq, double ∗tdecay, double ∗A, double ∗C)

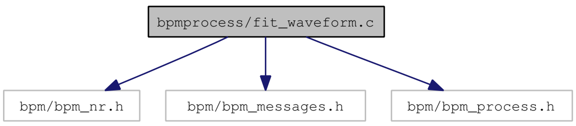
8.68 bpmprocess/fit_waveform.c File Reference

8.68.1 Detailed Description

Definition in file **fit_waveform.c**.

```
#include <bpm/bpm_nr.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** (**doublewf_t** *w, double t0, double i_freq, double i_tdecay, double i_amp, double i_phase, double *freq, double *tdecay, double *amp, double *phase)

8.69 bpmprocess/get_IQ.c File Reference

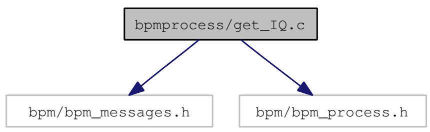
8.69.1 Detailed Description

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)

8.70 bpmprocess/get_pedestal.c File Reference

8.70.1 Detailed Description

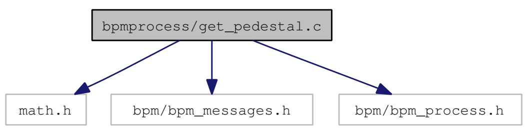
Definition in file **get_pedestal.c**.

`#include <math.h>`

`#include <bpm/bpm_messages.h>`

`#include <bpm/bpm_process.h>`

Include dependency graph for get_pedestal.c:



Functions

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

8.71 bpmprocess/get_pos.c File Reference

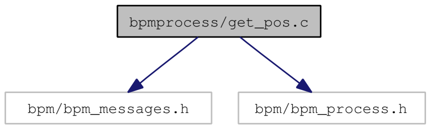
8.71.1 Detailed Description

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)

8.72 bpmprocess/get_slope.c File Reference

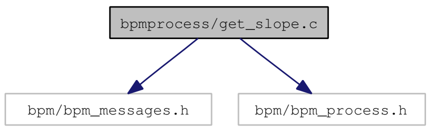
8.72.1 Detailed Description

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)

8.73 bpmprocess/get_t0.c File Reference

8.73.1 Detailed Description

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

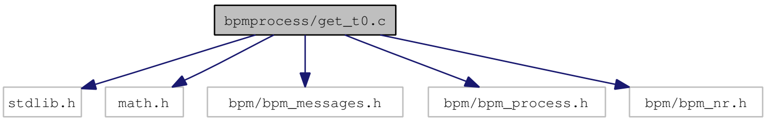
Definition in file **get_t0.c**.

`#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** (double ∗wf, double ped, int peak_sample, double peak_value, double peak_-fraction, int ∗start_sample)
- void **_find_t0_endfit** (double ∗wf, double ped, int peak_sample, double peak_value, double peak_-fraction, int ∗end_sample)
- int **get_t0** (**doublewf_t** ∗signal, double ∗t0)

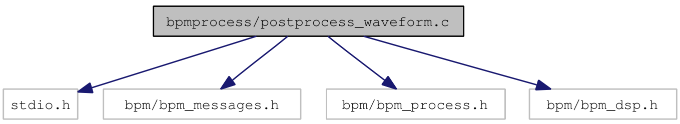
8.74 bpmprocess/postprocess_waveform.c File Reference

8.74.1 Detailed Description

Definition in file **postprocess_waveform.c**.

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

Include dependency graph for postprocess_waveform.c:



Functions

- int **postprocess_waveform** (**bpmconf_t** ∗bpm, **bpmproc_t** ∗proc, **bpmcalib_t** ∗cal, **bpmproc_t** ∗ampref, **bpmproc_t** ∗phaseref, unsigned int mode)

8.75 bpmprocess/process_caltone.c File Reference

8.75.1 Detailed Description

Definition in file **process_caltone.c**.

#include <stdio.h>

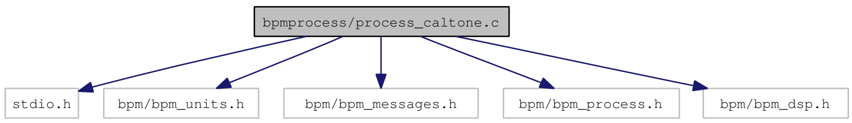
#include <bpm/bpm_units.h>

#include <bpm/bpm_messages.h>

#include <bpm/bpm_process.h>

#include <bpm/bpm_dsp.h>

Include dependency graph for process_caltone.c:



Functions

- int **process_caltone** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmproc_t** ∗proc, unsigned int mode)

8.76 bpmprocess/process_diode.c File Reference

8.76.1 Detailed Description

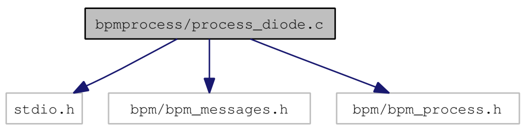
Definition in file **process_diode.c**.

#include <stdio.h>

#include <bpm/bpm_messages.h>

#include <bpm/bpm_process.h>

Include dependency graph for process_diode.c:



Functions

- int **process_diode** (**doublewf_t** ∗signal, **bpmconf_t** ∗conf, **bpmproc_t** ∗proc)

8.77 bpmprocess/process_dipole.c File Reference

8.77.1 Detailed Description

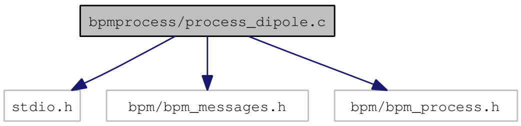
Definition in file **process_dipole.c**.

#include <stdio.h>

#include <bpm/bpm_messages.h>

#include <bpm/bpm_process.h>

Include dependency graph for process_dipole.c:



Functions

- int **process_dipole** (**doublewf_t** ∗sig, **bpmconf_t** ∗bpm, **bpmcalib_t** ∗cal, **bpmproc_t** ∗proc, **bpmproc_t** ∗trig, **bpmproc_t** ∗ampref, **bpmproc_t** ∗phaseref, unsigned int mode)

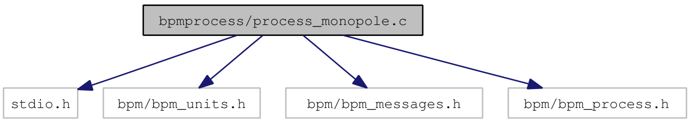
8.78 bpmprocess/process_monopole.c File Reference

8.78.1 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** (**doublewf_t** ∗sig, **bpmconf_t** ∗bpm, **bpmcalib_t** ∗cal, **bpmproc_t** ∗proc, **bpmproc_t** ∗trig, unsigned int mode)

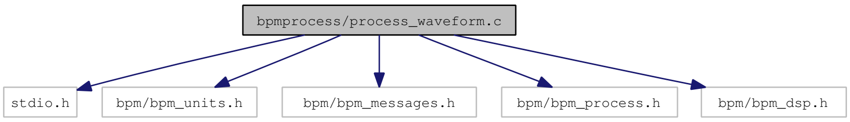
8.79 bpmprocess/process_waveform.c File Reference

8.79.1 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 <bpm/bpm_dsp.h>
```

Include dependency graph for process_waveform.c:



Functions

- int **process_waveform** (**doublewf_t** ∗signal, **bpmconf_t** ∗bpm, **bpmproc_t** ∗proc, **bpmproc_t** ∗trig, unsigned int mode)

8.80 bpmrf/bpm_rf.h File Reference

8.80.1 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**.

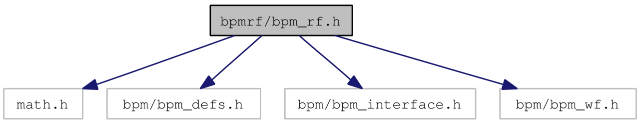
`#include <math.h>`

`#include <bpm/bpm_defs.h>`

`#include <bpm/bpm_interface.h>`

`#include <bpm/bpm_wf.h>`

Include dependency graph for bpm_rf.h:



Functions

- EXTERN int **rf_setup** (int nsamples, double sfreq)
- EXTERN int **rf_rectify** (**doublewf_t** ∗D, **complexwf_t** ∗RF)
- EXTERN int **rf_addLO** (double amp, double lofreq, enum **bpmphase_t** type, double phase, double phasenoise, **doublewf_t** ∗LO)
- EXTERN int **rf_mixer** (**doublewf_t** ∗RF_Re, **doublewf_t** ∗LO, **doublewf_t** ∗IF)
- EXTERN int **rf_amplify** (**doublewf_t** ∗RF, double dB)
- EXTERN int **rf_amplify_complex** (**complexwf_t** ∗RF, double dB)
- EXTERN int **rf_phase_shifter** (**complexwf_t** ∗RF, double rotation)

Variables

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

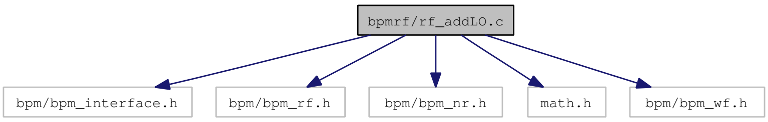
8.81 bpmrf/rf_addLO.c File Reference

8.81.1 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 <bpm/bpm_wf.h>
```

Include dependency graph for rf_addLO.c:



Functions

- int **rf_addLO** (double amp, double lofreq, enum **bpmphase_t** type, double phase, double phasenoise, **doublewf_t** ∗LO)

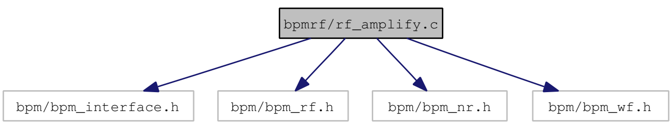
8.82 bpmrf/rf_amplify.c File Reference

8.82.1 Detailed Description

Definition in file **rf_amplify.c**.

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

Include dependency graph for rf_amplify.c:



Functions

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

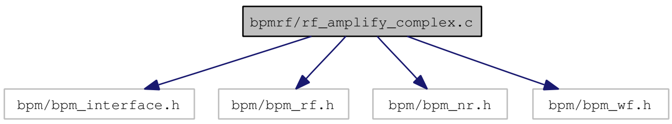
8.83 bpmrf/rf_amplify_complex.c File Reference

8.83.1 Detailed Description

Definition in file **rf_amplify_complex.c**.

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

Include dependency graph for rf_amplify_complex.c:



Functions

- int **rf_amplify_complex** (**complexwf_t** ∗RF, double dB)

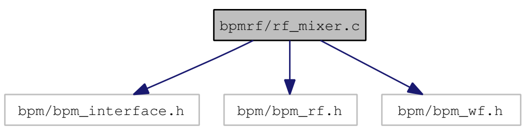
8.84 bpmrf/rf_mixer.c File Reference

8.84.1 Detailed Description

Definition in file **rf_mixer.c**.

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

Include dependency graph for rf_mixer.c:



Functions

- int **rf_mixer** (**doublewf_t** ∗RF, **doublewf_t** ∗LO, **doublewf_t** ∗IF)

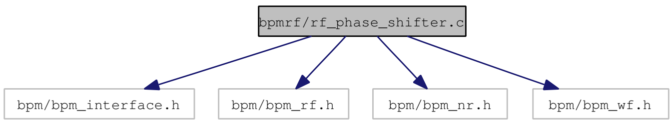
8.85 bpmrf/rf_phase_shifter.c File Reference

8.85.1 Detailed Description

Definition in file **rf_phase_shifter.c**.

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

Include dependency graph for rf_phase_shifter.c:



Functions

- int **rf_phase_shifter** (**complexwf_t** ∗RF, double rotation)

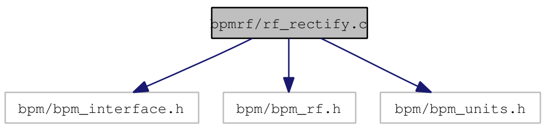
8.86 bpmrf/rf_rectify.c File Reference

8.86.1 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** (**doublewf_t** ∗D, **complexwf_t** ∗RF)

8.87 bpmrf/rf_setup.c File Reference

8.87.1 Detailed Description

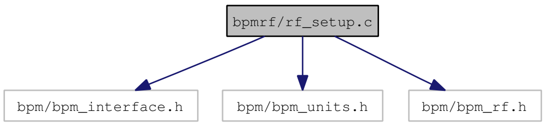
Definition in file **rf_setup.c**.

#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**

8.88 bpmsimulation/add_mode_response.c File Reference

8.88.1 Detailed Description

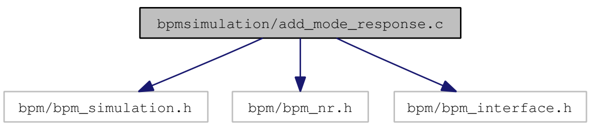
Definition in file **add_mode_response.c**.

#include <bpm/bpm_simulation.h>

#include <bpm/bpm_nr.h>

#include <bpm/bpm_interface.h>

Include dependency graph for add_mode_response.c:



Functions

- int **add_mode_response** (**bpmconf_t** ∗bpm, **bpmmode_t** ∗mode, **bunchconf_t** ∗bunch, **doublewf_t** ∗rf)

8.89 bpmsimulation/bpm_simulation.h File Reference

8.89.1 Detailed Description

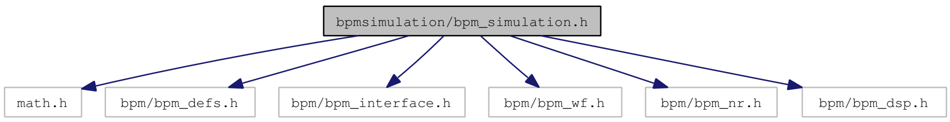
libbpm waveform simulation routines

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

Include dependency graph for bpm_simulation.h:



Defines

- #define **K_SAMPLE**
- #define **MODE_DECAY**
- #define **MODE_MAX_SAMPLES**

Functions

- EXTERN int **set_temp** (double TK)
- EXTERN int **set_time** (double ts)
- EXTERN int **generate_bpmsignal** (**bpmconf_t** ∗bpm, **bpmmode_t** ∗mode, **beamconf_t** ∗beam, **doublewf_t** ∗rf)
- EXTERN int **add_mode_response** (**bpmconf_t** ∗bpm, **bpmmode_t** ∗mode, **bunchconf_t** ∗bunch, **doublewf_t** ∗rf)
- EXTERN **complex_t get_mode_amplitude** (**bpmconf_t** ∗bpm, **bpmmode_t** ∗mode, **bunchconf_t** ∗bunch)
- EXTERN **doublewf_t** ∗ **generate_diodesignal** (**doublewf_t** ∗rf, double sens, **filter_t** ∗filt, triggertype_t diode)
- EXTERN int **get_mode_response** (**bpmmode_t** ∗mode)
- EXTERN int **digitise** (**doublewf_t** ∗IF, int nbits, double range_min, double range_max, double clock_jitter, double digi_noise, unsigned int ipmode, **intwf_t** ∗wf)

Variables

- EXTERN double **ambient_temp**
- EXTERN double **system_time**

8.90 bpmsimulation/digitise.c File Reference

8.90.1 Detailed Description

Definition in file **digitise.c**.

#include <bpm/bpm_messages.h>

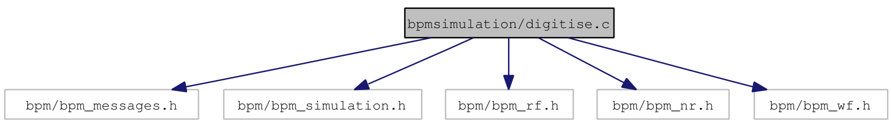
#include <bpm/bpm_simulation.h>

#include <bpm/bpm_rf.h>

#include <bpm/bpm_nr.h>

#include <bpm/bpm_wf.h>

Include dependency graph for digitise.c:



Functions

- int **digitise** (**doublewf_t** ∗IF, int nbits, double range_min, double range_max, double clock_jitter, double digi_noise, unsigned int ipmode, **intwf_t** ∗wf)

8.91 bpmsimulation/generate_bpmsignal.c File Reference

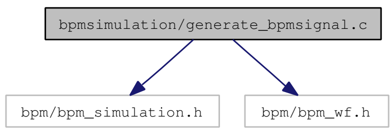
8.91.1 Detailed Description

Definition in file **generate_bpmsignal.c**.

#include <bpm/bpm_simulation.h>

#include <bpm/bpm_wf.h>

Include dependency graph for generate_bpmsignal.c:



Functions

- int **generate_bpmsignal** (**bpmconf_t** ∗bpm, **bpmmode_t** ∗mode, **beamconf_t** ∗beam, **doublewf_t** ∗rf)

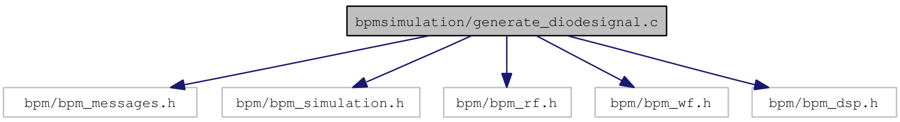
8.92 bpmsimulation/generate_diodesignal.c File Reference

8.92.1 Detailed Description

Definition in file **generate_diodesignal.c**.

```
#include <bpm/bpm_messages.h>
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_rf.h>
#include <bpm/bpm_wf.h>
#include <bpm/bpm_dsp.h>
```

Include dependency graph for generate_diodesignal.c:



Functions

- **doublewf_t** ∗ **generate_diodesignal** (**doublewf_t** ∗rf, double sens, **filter_t** ∗filt, triggertype_t diode)

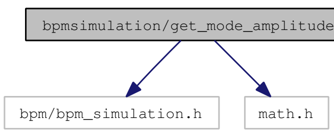
8.93 bpmsimulation/get_mode_amplitude.c File Reference

8.93.1 Detailed Description

Definition in file **get_mode_amplitude.c**.

```
#include <bpm/bpm_simulation.h>
#include <math.h>
```

Include dependency graph for get_mode_amplitude.c:



Functions

- **complex_t get_mode_amplitude** (**bpmconf_t** ∗bpm, **bpmmode_t** ∗mode, **bunchconf_t** ∗bunch)

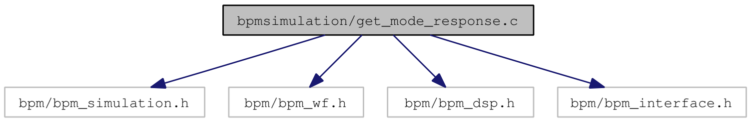
8.94 bpmsimulation/get_mode_response.c File Reference

8.94.1 Detailed Description

Definition in file **get_mode_response.c**.

```
#include <bpm/bpm_simulation.h>
#include <bpm/bpm_wf.h>
#include <bpm/bpm_dsp.h>
#include <bpm/bpm_interface.h>
```

Include dependency graph for get_mode_response.c:



Functions

- int **get_mode_response** (**bpmmode_t** ∗mode)

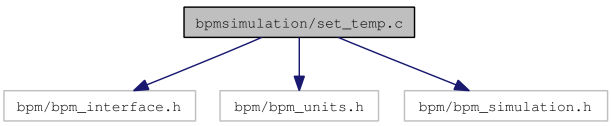
8.95 bpmsimulation/set_temp.c File Reference

8.95.1 Detailed Description

Definition in file **set_temp.c**.

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

Include dependency graph for set_temp.c:



Functions

- int **set_temp** (double TK)

Variables

- double **ambient_temp**

8.96 bpmsimulation/set_time.c File Reference

8.96.1 Detailed Description

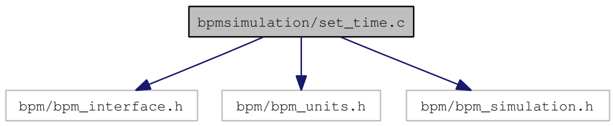
Definition in file **set_time.c**.

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

`#include <bpm/bpm_units.h>`

`#include <bpm/bpm_simulation.h>`

Include dependency graph for set_time.c:



Functions

- int **set_time** (double ts)

Variables

- double **system_time**

8.97 bpmwf/bpm_wf.h File Reference

8.97.1 Detailed Description

Simple waveform handling routines for libbpm.

Definition in file **bpm_wf.h**.

`#include <math.h>`

`#include <float.h>`

`#include <stdio.h>`

`#include <stdlib.h>`

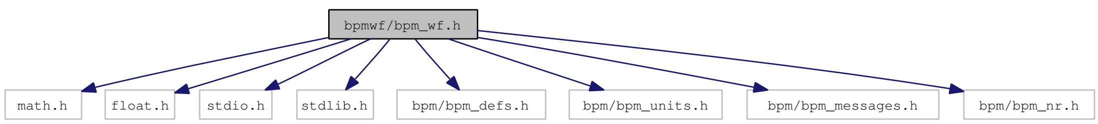
`#include "bpm/bpm_defs.h"`

`#include "bpm/bpm_units.h"`

`#include "bpm/bpm_messages.h"`

`#include "bpm/bpm_nr.h"`

Include dependency graph for bpm_wf.h:



Data Structures

- struct **doublewf_t**
- struct **intwf_t**
- struct **complexwf_t**
- struct **wfstat_t**

Defines

- #define **WF_EPS**
- #define **MAX_ALLOWED_NS**
- #define **WF_NEAREST**
- #define **WF_LINEAR**
- #define **WF_QUADRATIC**
- #define **WF_SINC**
- #define **WF_LANCZOS**

Functions

- EXTERN int **wfstat_reset** (**wfstat_t** ∗s)
- EXTERN void **wfstat_print** (FILE ∗of, **wfstat_t** ∗s)
- EXTERN **doublewf_t** ∗ **doublewf** (int ns, double fs)
- EXTERN **doublewf_t** ∗ **doublewf_time_series** (int ns, double fs)
- EXTERN **doublewf_t** ∗ **doublewf_sample_series** (int ns, double fs)
- EXTERN **doublewf_t** ∗ **doublewf_frequency_series** (int ns, double fs)
- EXTERN int **doublewf_setvalues** (**doublewf_t** ∗w, double ∗x)
- EXTERN int **doublewf_setfunction** (**doublewf_t** ∗w, double(∗wffun)(double t, int, double ∗), int npars, double ∗par)
- EXTERN int **doublewf_copy** (**doublewf_t** ∗copy, **doublewf_t** ∗src)
- EXTERN **doublewf_t** ∗ **doublewf_copy_new** (**doublewf_t** ∗w)
- EXTERN int **doublewf_subset** (**doublewf_t** ∗sub, **doublewf_t** ∗w, int i1, int i2)
- EXTERN int **doublewf_reset** (**doublewf_t** ∗w)
- EXTERN void **doublewf_delete** (**doublewf_t** ∗w)
- EXTERN **intwf_t** ∗ **intwf_cast_new** (**doublewf_t** ∗w)
- EXTERN int **intwf_cast** (**intwf_t** ∗iw, **doublewf_t** ∗w)
- EXTERN int **doublewf_compat** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_add** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_subtract** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_multiply** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_divide** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- EXTERN int **doublewf_scale** (double f, **doublewf_t** ∗w)
- EXTERN int **doublewf_bias** (double c, **doublewf_t** ∗w)
- EXTERN int **doublewf_add_cwtone** (**doublewf_t** ∗w, double amp, double phase, double freq, double phasenoise)
- EXTERN int **doublewf_add_dcywave** (**doublewf_t** ∗w, double amp, double phase, double freq, double ttrig, double tdcy, double phasenoise)
- EXTERN int **doublewf_add_ampnoise** (**doublewf_t** ∗w, double sigma)
- EXTERN int **doublewf_basic_stats** (**doublewf_t** ∗w, int s0, int s1, **wfstat_t** ∗stats)
- EXTERN int **doublewf_derive** (**doublewf_t** ∗w)
- EXTERN int **doublewf_integrate** (**doublewf_t** ∗w)
- EXTERN void **doublewf_print** (FILE ∗of, **doublewf_t** ∗w)
- EXTERN double **doublewf_getvalue** (**doublewf_t** ∗w, double t, unsigned int mode)
- EXTERN int **doublewf_resample** (**doublewf_t** ∗w2, double fs, **doublewf_t** ∗w1, unsigned int mode)
- EXTERN **intwf_t** ∗ **intwf** (int ns, double fs)
- EXTERN **intwf_t** ∗ **intwf_sample_series** (int ns, double fs)
- EXTERN int **intwf_setvalues** (**intwf_t** ∗w, int ∗x)

- EXTERN int **intwf_setfunction** (**intwf_t** ∗w, int(∗wffun)(double t, int, double ∗), int npars, double ∗par)
- EXTERN int **intwf_copy** (**intwf_t** ∗copy, **intwf_t** ∗src)
- EXTERN **intwf_t** ∗ **intwf_copy_new** (**intwf_t** ∗w)
- EXTERN int **intwf_subset** (**intwf_t** ∗sub, **intwf_t** ∗w, int i1, int i2)
- EXTERN int **intwf_reset** (**intwf_t** ∗w)
- EXTERN void **intwf_delete** (**intwf_t** ∗w)
- EXTERN **doublewf_t** ∗ **doublewf_cast_new** (**intwf_t** ∗w)
- EXTERN int **doublewf_cast** (**doublewf_t** ∗w, **intwf_t** ∗iw)
- EXTERN int **intwf_compat** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_add** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_subtract** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_multiply** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_divide** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- EXTERN int **intwf_scale** (int f, **intwf_t** ∗w)
- EXTERN int **intwf_bias** (int c, **intwf_t** ∗w)
- EXTERN int **intwf_add_cwtone** (**intwf_t** ∗w, double amp, double phase, double freq, double phasenoise)
- EXTERN int **intwf_add_dcywave** (**intwf_t** ∗w, double amp, double phase, double freq, double ttrig, double tdcy, double phasenoise)
- EXTERN int **intwf_add_ampnoise** (**intwf_t** ∗w, double sigma)
- EXTERN int **intwf_basic_stats** (**intwf_t** ∗w, int s0, int s1, **wfstat_t** ∗stats)
- EXTERN int **intwf_derive** (**intwf_t** ∗w)
- EXTERN int **intwf_integrate** (**intwf_t** ∗w)
- EXTERN void **intwf_print** (FILE ∗of, **intwf_t** ∗w)
- EXTERN int **intwf_getvalue** (**intwf_t** ∗w, double t, unsigned int mode)
- EXTERN int **intwf_resample** (**intwf_t** ∗w2, double fs, **intwf_t** ∗w1, unsigned int mode)
- EXTERN **complexwf_t** ∗ **complexwf** (int ns, double fs)
- EXTERN **complexwf_t** ∗ **complexwf_copy_new** (**complexwf_t** ∗w)
- EXTERN int **complexwf_copy** (**complexwf_t** ∗copy, **complexwf_t** ∗src)
- EXTERN int **complexwf_subset** (**complexwf_t** ∗sub, **complexwf_t** ∗w, int i1, int i2)
- EXTERN int **complexwf_setvalues** (**complexwf_t** ∗w, **complex_t** ∗x)
- EXTERN int **complexwf_setfunction** (**complexwf_t** ∗w, **complex_t**(∗wffun)(double, int, double ∗), int npars, double ∗par)
- EXTERN int **complexwf_reset** (**complexwf_t** ∗w)
- EXTERN void **complexwf_delete** (**complexwf_t** ∗w)
- EXTERN int **complexwf_compat** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_add** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_subtract** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_multiply** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_divide** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- EXTERN int **complexwf_scale** (**complex_t** f, **complexwf_t** ∗w)
- EXTERN int **complexwf_bias** (**complex_t** c, **complexwf_t** ∗w)
- EXTERN int **complexwf_add_cwtone** (**complexwf_t** ∗w, double amp, double phase, double freq, double phasenoise)
- EXTERN int **complexwf_add_dcywave** (**complexwf_t** ∗w, double amp, double phase, double freq, double ttrig, double tdcy, double phasenoise)
- EXTERN int **complexwf_add_noise** (**complexwf_t** ∗w, double sigma)
- EXTERN int **complexwf_add_ampnoise** (**complexwf_t** ∗w, double sigma)
- EXTERN int **complexwf_add_phasenoise** (**complexwf_t** ∗w, double sigma)

- EXTERN void **complexwf_print** (FILE ∗of, **complexwf_t** ∗w)
- EXTERN int **complexwf_getreal** (**doublewf_t** ∗re, **complexwf_t** ∗z)
- EXTERN int **complexwf_getimag** (**doublewf_t** ∗im, **complexwf_t** ∗z)
- EXTERN int **complexwf_getamp** (**doublewf_t** ∗r, **complexwf_t** ∗z)
- EXTERN int **complexwf_getphase** (**doublewf_t** ∗theta, **complexwf_t** ∗z)
- EXTERN **doublewf_t** ∗ **complexwf_getreal_new** (**complexwf_t** ∗z)
- EXTERN **doublewf_t** ∗ **complexwf_getimag_new** (**complexwf_t** ∗z)
- EXTERN **doublewf_t** ∗ **complexwf_getamp_new** (**complexwf_t** ∗z)
- EXTERN **doublewf_t** ∗ **complexwf_getphase_new** (**complexwf_t** ∗z)
- EXTERN int **complexwf_setreal** (**complexwf_t** ∗z, **doublewf_t** ∗re)
- EXTERN int **complexwf_setimag** (**complexwf_t** ∗z, **doublewf_t** ∗im)
- EXTERN int **time_to_sample** (double fs, int ns, double t, int ∗iS)
- EXTERN int **freq_to_sample** (double fs, int ns, double f, int ∗iS)
- EXTERN int **sample_to_time** (double fs, int ns, int iS, double ∗t)
- EXTERN int **sample_to_freq** (double fs, int ns, int iS, double ∗f)

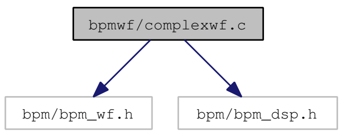
8.98 bpmwf/complexwf.c File Reference

8.98.1 Detailed Description

Definition in file **complexwf.c**.

```
#include <bpm/bpm_wf.h>
#include <bpm/bpm_dsp.h>
```

Include dependency graph for complexwf.c:



Functions

- **complexwf_t** ∗ **complexwf** (int ns, double fs)
- **complexwf_t** ∗ **complexwf_copy_new** (**complexwf_t** ∗w)
- int **complexwf_copy** (**complexwf_t** ∗copy, **complexwf_t** ∗src)
- int **complexwf_subset** (**complexwf_t** ∗sub, **complexwf_t** ∗w, int i1, int i2)
- int **complexwf_setvalues** (**complexwf_t** ∗w, **complex_t** ∗x)
- int **complexwf_setfunction** (**complexwf_t** ∗w, **complex_t**(∗wffun)(double, int, double ∗), int npars, double ∗par)
- int **complexwf_reset** (**complexwf_t** ∗w)
- void **complexwf_delete** (**complexwf_t** ∗w)
- int **complexwf_compat** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- int **complexwf_add** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- int **complexwf_subtract** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- int **complexwf_multiply** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- int **complexwf_divide** (**complexwf_t** ∗w1, **complexwf_t** ∗w2)
- int **complexwf_scale** (**complex_t** f, **complexwf_t** ∗w)

- int **complexwf_bias** (**complex_t** c, **complexwf_t** *w)
- int **complexwf_add_cwtone** (**complexwf_t** *w, double amp, double phase, double freq, double phasenoise)
- int **complexwf_add_dcywave** (**complexwf_t** *w, double amp, double phase, double freq, double ttrig, double tdcy, double phasenoise)
- int **complexwf_add_noise** (**complexwf_t** *w, double sigma)
- int **complexwf_add_ampnoise** (**complexwf_t** *w, double sigma)
- int **complexwf_add_phasenoise** (**complexwf_t** *w, double sigma)
- void **complexwf_print** (FILE *of, **complexwf_t** *w)
- int **complexwf_getreal** (**doublewf_t** *re, **complexwf_t** *z)
- int **complexwf_getimag** (**doublewf_t** *im, **complexwf_t** *z)
- int **complexwf_getamp** (**doublewf_t** *r, **complexwf_t** *z)
- int **complexwf_getphase** (**doublewf_t** *theta, **complexwf_t** *z)
- int **complexwf_setreal** (**complexwf_t** *z, **doublewf_t** *re)
- int **complexwf_setimag** (**complexwf_t** *z, **doublewf_t** *im)
- **doublewf_t** * **complexwf_getreal_new** (**complexwf_t** *z)
- **doublewf_t** * **complexwf_getimag_new** (**complexwf_t** *z)
- **doublewf_t** * **complexwf_getamp_new** (**complexwf_t** *z)
- **doublewf_t** * **complexwf_getphase_new** (**complexwf_t** *z)

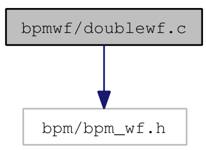
8.99 bpmwf/doublewf.c File Reference

8.99.1 Detailed Description

Definition in file **doublewf.c**.

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

Include dependency graph for doublewf.c:



Functions

- **doublewf_t** * **doublewf** (int ns, double fs)
- **doublewf_t** * **doublewf_sample_series** (int ns, double fs)
- **doublewf_t** * **doublewf_time_series** (int ns, double fs)
- **doublewf_t** * **doublewf_frequency_series** (int ns, double fs)
- **doublewf_t** * **doublewf_copy_new** (**doublewf_t** *w)
- int **doublewf_copy** (**doublewf_t** *copy, **doublewf_t** *src)
- int **doublewf_subset** (**doublewf_t** *sub, **doublewf_t** *w, int i1, int i2)
- int **doublewf_setvalues** (**doublewf_t** *w, double *x)
- int **doublewf_setfunction** (**doublewf_t** *w, double(*wffun)(double, int, double *), int npars, double *par)
- int **doublewf_reset** (**doublewf_t** *w)
- void **doublewf_delete** (**doublewf_t** *w)

- **intwf_t** ∗ **intwf_cast_new** (**doublewf_t** ∗w)
- int **intwf_cast** (**intwf_t** ∗iw, **doublewf_t** ∗w)
- int **doublewf_compat** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- int **doublewf_add** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- int **doublewf_subtract** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- int **doublewf_multiply** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- int **doublewf_divide** (**doublewf_t** ∗w1, **doublewf_t** ∗w2)
- int **doublewf_scale** (double f, **doublewf_t** ∗w)
- int **doublewf_bias** (double c, **doublewf_t** ∗w)
- int **doublewf_add_cwtone** (**doublewf_t** ∗w, double amp, double phase, double freq, double phasenoise)
- int **doublewf_add_dcywave** (**doublewf_t** ∗w, double amp, double phase, double freq, double ttrig, double tdcy, double phasenoise)
- int **doublewf_add_ampnoise** (**doublewf_t** ∗w, double sigma)
- int **doublewf_basic_stats** (**doublewf_t** ∗w, int s0, int s1, **wfstat_t** ∗stats)
- int **doublewf_derive** (**doublewf_t** ∗w)
- int **doublewf_integrate** (**doublewf_t** ∗w)
- void **doublewf_print** (FILE ∗of, **doublewf_t** ∗w)
- double **doublewf_getvalue** (**doublewf_t** ∗w, double t, unsigned int mode)
- int **doublewf_resample** (**doublewf_t** ∗w2, double fs, **doublewf_t** ∗w1, unsigned int mode)

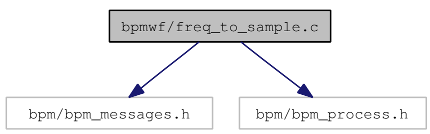
8.100 bpmwf/freq_to_sample.c File Reference

8.100.1 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)

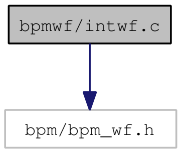
8.101 bpmwf/intwf.c File Reference

8.101.1 Detailed Description

Definition in file **intwf.c**.

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

Include dependency graph for intwf.c:



Functions

- **intwf_t** ∗ **intwf** (int ns, double fs)
- **intwf_t** ∗ **intwf_sample_series** (int ns, double fs)
- **intwf_t** ∗ **intwf_copy_new** (**intwf_t** ∗w)
- int **intwf_copy** (**intwf_t** ∗copy, **intwf_t** ∗src)
- int **intwf_subset** (**intwf_t** ∗sub, **intwf_t** ∗w, int i1, int i2)
- int **intwf_setvalues** (**intwf_t** ∗w, int ∗x)
- int **intwf_setfunction** (**intwf_t** ∗w, int(∗wffun)(double, int, double ∗), int npars, double ∗par)
- int **intwf_reset** (**intwf_t** ∗w)
- void **intwf_delete** (**intwf_t** ∗w)
- **doublewf_t** ∗ **doublewf_cast_new** (**intwf_t** ∗iw)
- int **doublewf_cast** (**doublewf_t** ∗w, **intwf_t** ∗iw)
- int **intwf_compat** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- int **intwf_add** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- int **intwf_subtract** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- int **intwf_multiply** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- int **intwf_divide** (**intwf_t** ∗w1, **intwf_t** ∗w2)
- int **intwf_scale** (int f, **intwf_t** ∗w)
- int **intwf_bias** (int c, **intwf_t** ∗w)
- int **intwf_add_cwtone** (**intwf_t** ∗w, double amp, double phase, double freq, double phasenoise)
- int **intwf_add_dcywave** (**intwf_t** ∗w, double amp, double phase, double freq, double ttrig, double tdcy, double phasenoise)
- int **intwf_add_ampnoise** (**intwf_t** ∗w, double sigma)
- int **intwf_basic_stats** (**intwf_t** ∗w, int s0, int s1, **wfstat_t** ∗stats)
- int **intwf_derive** (**intwf_t** ∗w)
- int **intwf_integrate** (**intwf_t** ∗w)
- void **intwf_print** (FILE ∗of, **intwf_t** ∗w)
- int **intwf_getvalue** (**intwf_t** ∗w, double t, unsigned int mode)
- int **intwf_resample** (**intwf_t** ∗w2, double fs, **intwf_t** ∗w1, unsigned int mode)

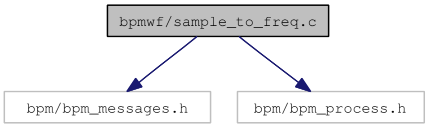
8.102 bpmwf/sample_to_freq.c File Reference

8.102.1 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)

8.103 bpmwf/sample_to_time.c File Reference

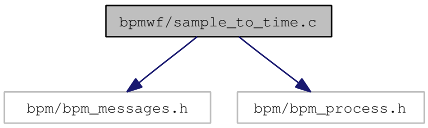
8.103.1 Detailed Description

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)

8.104 bpmwf/time_to_sample.c File Reference

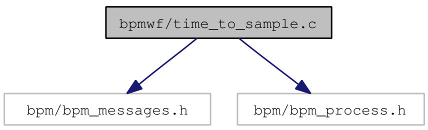
8.104.1 Detailed Description

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)

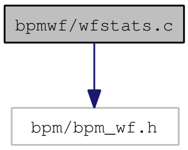
8.105 bpmwf/wfstats.c File Reference

8.105.1 Detailed Description

Definition in file **wfstats.c**.

`#include <bpm/bpm_wf.h>`

Include dependency graph for wfstats.c:



Functions

- int **wfstat_reset** (**wfstat_t** ∗s)
- void **wfstat_print** (FILE ∗of, **wfstat_t** ∗s)

Index