Previous topic
gnuradio.audio
Next topic
gnuradio.blocks
This Page
Show Source
Quick search
Go
Enter search terms or a module, class or function name.

gnuradio.analog

Blocks and utilities for analog modulation and demodulation.

gnuradio.analog.**agc2_cc**(*float attack_rate=1e-1*, *float decay_rate=1e-2*, *float reference=1.0*, *float gain=1.0*) → agc2_cc_sptr

high performance Automatic Gain Control class with attack and decay rates.

For Power the absolute value of the complex number is used.

Constructor Specific Documentation:

Build a complex value AGC loop block with attack and decay rates.

Parameters:
- **attack_rate** – the update rate of the loop when in attack mode.
- **decay_rate** – the update rate of the loop when in decay mode.
- **reference** – reference value to adjust signal power to.
- **gain** – initial gain value.

agc2_cc_sptr.**active_thread_priority**(*agc2_cc_sptr self*) → int

agc2_cc_sptr.**attack_rate**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**decay_rate**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**declare_sample_delay**(*agc2_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(agc2_cc_sptr self, unsigned int delay)

agc2_cc_sptr.**gain**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**max_gain**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**message_subscribers**(*agc2_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

agc2_cc_sptr.**min_noutput_items**(*agc2_cc_sptr self*) → int

agc2_cc_sptr.**pc_input_buffers_full_avg**(*agc2_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(agc2_cc_sptr self) -> pmt_vector_float

agc2_cc_sptr.**pc_noutput_items_avg**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**pc_nproduced_avg**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**pc_output_buffers_full_avg**(*agc2_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(agc2_cc_sptr self) -> pmt_vector_float

agc2_cc_sptr.**pc_throughput_avg**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**pc_work_time_avg**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**pc_work_time_total**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**reference**(*agc2_cc_sptr self*) → float

agc2_cc_sptr.**sample_delay**(*agc2_cc_sptr self*, *int which*) → unsigned int

agc2_cc_sptr.**set_attack_rate**(*agc2_cc_sptr self*, *float rate*)

agc2_cc_sptr.**set_decay_rate**(*agc2_cc_sptr self*, *float rate*)

agc2_cc_sptr.**set_gain**(*agc2_cc_sptr self*, *float gain*)

agc2_cc_sptr.**set_max_gain**(*agc2_cc_sptr self*, *float max_gain*)

agc2_cc_sptr.**set_min_noutput_items**(*agc2_cc_sptr self*, *int m*)

agc2_cc_sptr.**set_reference**(*agc2_cc_sptr self*, *float reference*)

agc2_cc_sptr.**set_thread_priority**(*agc2_cc_sptr self*, *int priority*) → int

agc2_cc_sptr.**thread_priority**(*agc2_cc_sptr self*) → int

gnuradio.analog.**agc2_ff**(*float attack_rate=1e-1*, *float decay_rate=1e-2*, *float reference=1.0*, *float gain=1.0*) → agc2_ff_sptr

high performance Automatic Gain Control class with attack and decay rates.

Power is approximated by absolute value

Constructor Specific Documentation:

Build a floating point AGC loop block with attack and decay rates.

Parameters:
- **attack_rate** – the update rate of the loop when in attack mode.
- **decay_rate** – the update rate of the loop when in decay mode.
- **reference** – reference value to adjust signal power to.
- **gain** – initial gain value.

agc2_ff_sptr.**active_thread_priority**(*agc2_ff_sptr self*) → int

agc2_ff_sptr.**attack_rate**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**decay_rate**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**declare_sample_delay**(*agc2_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(agc2_ff_sptr self, unsigned int delay)

agc2_ff_sptr.**gain**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**max_gain**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**message_subscribers**(*agc2_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

agc2_ff_sptr.**min_noutput_items**(*agc2_ff_sptr self*) → int

agc2_ff_sptr.**pc_input_buffers_full_avg**(*agc2_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(agc2_ff_sptr self) -> pmt_vector_float

agc2_ff_sptr.**pc_noutput_items_avg**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**pc_nproduced_avg**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**pc_output_buffers_full_avg**(*agc2_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(agc2_ff_sptr self) -> pmt_vector_float

agc2_ff_sptr.**pc_throughput_avg**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**pc_work_time_avg**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**pc_work_time_total**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**reference**(*agc2_ff_sptr self*) → float

agc2_ff_sptr.**sample_delay**(*agc2_ff_sptr self*, *int which*) → unsigned int

agc2_ff_sptr.**set_attack_rate**(*agc2_ff_sptr self*, *float rate*)

agc2_ff_sptr.**set_decay_rate**(*agc2_ff_sptr self*, *float rate*)

agc2_ff_sptr.**set_gain**(*agc2_ff_sptr self*, *float gain*)

agc2_ff_sptr.**set_max_gain**(*agc2_ff_sptr self*, *float max_gain*)

agc2_ff_sptr.**set_min_noutput_items**(*agc2_ff_sptr self*, *int m*)

agc2_ff_sptr.**set_reference**(*agc2_ff_sptr self*, *float reference*)

agc2_ff_sptr.**set_thread_priority**(*agc2_ff_sptr self*, *int priority*) → int

agc2_ff_sptr.**thread_priority**(*agc2_ff_sptr self*) → int

gnuradio.analog.**agc3_cc**(*float attack_rate=1e-1*, *float decay_rate=1e-2*, *float reference=1.0*, *float gain=1.0*, *int iir_update_decim=1*) → agc3_cc_sptr

high performance Automatic Gain Control class with attack and decay rates.

Unlike the AGC2 loop, this uses an initial linear calculation at the beginning for very fast initial acquisition. Moves to IIR model for tracking purposes.

For Power the absolute value of the complex number is used.

Constructor Specific Documentation:

Build a complex value AGC loop block with attack and decay rates.

Parameters:
- **attack_rate** – the update rate of the loop when in attack mode.
- **decay_rate** – the update rate of the loop when in decay mode.
- **reference** – reference value to adjust signal power to.
- **gain** – initial gain value.
- **iir_update_decim** – stride by this number of samples before computing an IIR gain update

agc3_cc_sptr.**active_thread_priority**(*agc3_cc_sptr self*) → int

agc3_cc_sptr.**attack_rate**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**decay_rate**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**declare_sample_delay**(*agc3_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(agc3_cc_sptr self, unsigned int delay)

agc3_cc_sptr.**gain**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**max_gain**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**message_subscribers**(*agc3_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

agc3_cc_sptr.**min_noutput_items**(*agc3_cc_sptr self*) → int

agc3_cc_sptr.**pc_input_buffers_full_avg**(*agc3_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(agc3_cc_sptr self) -> pmt_vector_float

agc3_cc_sptr.**pc_noutput_items_avg**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**pc_nproduced_avg**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**pc_output_buffers_full_avg**(*agc3_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(agc3_cc_sptr self) -> pmt_vector_float

agc3_cc_sptr.**pc_throughput_avg**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**pc_work_time_avg**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**pc_work_time_total**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**reference**(*agc3_cc_sptr self*) → float

agc3_cc_sptr.**sample_delay**(*agc3_cc_sptr self*, *int which*) → unsigned int

agc3_cc_sptr.**set_attack_rate**(*agc3_cc_sptr self*, *float rate*)

agc3_cc_sptr.**set_decay_rate**(*agc3_cc_sptr self*, *float rate*)

agc3_cc_sptr.**set_gain**(*agc3_cc_sptr self*, *float gain*)

agc3_cc_sptr.**set_max_gain**(*agc3_cc_sptr self*, *float max_gain*)

agc3_cc_sptr.**set_min_noutput_items**(*agc3_cc_sptr self*, *int m*)

agc3_cc_sptr.**set_reference**(*agc3_cc_sptr self*, *float reference*)

agc3_cc_sptr.**set_thread_priority**(*agc3_cc_sptr self*, *int priority*) → int

agc3_cc_sptr.**thread_priority**(*agc3_cc_sptr self*) → int

gnuradio.analog.**agc_cc**(*float rate=1e-4*, *float reference=1.0*, *float gain=1.0*) → agc_cc_sptr

high performance Automatic Gain Control class

For Power the absolute value of the complex number is used.

Constructor Specific Documentation:

Build a complex value AGC loop block.

Parameters:
- **rate** – the update rate of the loop.
- **reference** – reference value to adjust signal power to.
- **gain** – initial gain value.

agc_cc_sptr.**active_thread_priority**(*agc_cc_sptr self*) → int

agc_cc_sptr.**declare_sample_delay**(*agc_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(agc_cc_sptr self, unsigned int delay)

agc_cc_sptr.**gain**(*agc_cc_sptr self*) → float

agc_cc_sptr.**max_gain**(*agc_cc_sptr self*) → float

agc_cc_sptr.**message_subscribers**(*agc_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

agc_cc_sptr.**min_noutput_items**(*agc_cc_sptr self*) → int

agc_cc_sptr.**pc_input_buffers_full_avg**(*agc_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(agc_cc_sptr self) -> pmt_vector_float

agc_cc_sptr.**pc_noutput_items_avg**(*agc_cc_sptr self*) → float

agc_cc_sptr.**pc_nproduced_avg**(*agc_cc_sptr self*) → float

agc_cc_sptr.**pc_output_buffers_full_avg**(*agc_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(agc_cc_sptr self) -> pmt_vector_float

agc_cc_sptr.**pc_throughput_avg**(*agc_cc_sptr self*) → float

agc_cc_sptr.**pc_work_time_avg**(*agc_cc_sptr self*) → float

agc_cc_sptr.**pc_work_time_total**(*agc_cc_sptr self*) → float

agc_cc_sptr.**rate**(*agc_cc_sptr self*) → float

agc_cc_sptr.**reference**(*agc_cc_sptr self*) → float

agc_cc_sptr.**sample_delay**(*agc_cc_sptr self*, *int which*) → unsigned int

agc_cc_sptr.**set_gain**(*agc_cc_sptr self*, *float gain*)

agc_cc_sptr.**set_max_gain**(*agc_cc_sptr self*, *float max_gain*)

agc_cc_sptr.**set_min_noutput_items**(*agc_cc_sptr self*, *int m*)

agc_cc_sptr.**set_rate**(*agc_cc_sptr self*, *float rate*)

agc_cc_sptr.**set_reference**(*agc_cc_sptr self*, *float reference*)

agc_cc_sptr.**set_thread_priority**(*agc_cc_sptr self*, *int priority*) → int

agc_cc_sptr.**thread_priority**(*agc_cc_sptr self*) → int

gnuradio.analog.**agc_ff**(*float rate=1e-4*, *float reference=1.0*, *float gain=1.0*) → agc_ff_sptr

high performance Automatic Gain Control class

Power is approximated by absolute value

Constructor Specific Documentation:

Build a floating point AGC loop block.

Parameters:
- **rate** – the update rate of the loop.
- **reference** – reference value to adjust signal power to.
- **gain** – initial gain value.

agc_ff_sptr.**active_thread_priority**(*agc_ff_sptr self*) → int

agc_ff_sptr.**declare_sample_delay**(*agc_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(agc_ff_sptr self, unsigned int delay)

agc_ff_sptr.**gain**(*agc_ff_sptr self*) → float

agc_ff_sptr.**max_gain**(*agc_ff_sptr self*) → float

agc_ff_sptr.**message_subscribers**(*agc_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

agc_ff_sptr.**min_noutput_items**(*agc_ff_sptr self*) → int

agc_ff_sptr.**pc_input_buffers_full_avg**(*agc_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(agc_ff_sptr self) -> pmt_vector_float

agc_ff_sptr.**pc_noutput_items_avg**(*agc_ff_sptr self*) → float

agc_ff_sptr.**pc_nproduced_avg**(*agc_ff_sptr self*) → float

agc_ff_sptr.**pc_output_buffers_full_avg**(*agc_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(agc_ff_sptr self) -> pmt_vector_float

agc_ff_sptr.**pc_throughput_avg**(*agc_ff_sptr self*) → float

agc_ff_sptr.**pc_work_time_avg**(*agc_ff_sptr self*) → float

agc_ff_sptr.**pc_work_time_total**(*agc_ff_sptr self*) → float

agc_ff_sptr.**rate**(*agc_ff_sptr self*) → float

agc_ff_sptr.**reference**(*agc_ff_sptr self*) → float

agc_ff_sptr.**sample_delay**(*agc_ff_sptr self*, *int which*) → unsigned int

agc_ff_sptr.**set_gain**(*agc_ff_sptr self*, *float gain*)

agc_ff_sptr.**set_max_gain**(*agc_ff_sptr self*, *float max_gain*)

agc_ff_sptr.**set_min_noutput_items**(*agc_ff_sptr self*, *int m*)

agc_ff_sptr.**set_rate**(*agc_ff_sptr self*, *float rate*)

agc_ff_sptr.**set_reference**(*agc_ff_sptr self*, *float reference*)

agc_ff_sptr.**set_thread_priority**(*agc_ff_sptr self*, *int priority*) → int

agc_ff_sptr.**thread_priority**(*agc_ff_sptr self*) → int

gnuradio.analog.**cpfsk_bc**(*float k*, *float ampl*, *int samples_per_sym*) → cpfsk_bc_sptr

Perform continuous phase 2-level frequency shift keying modulation on an input stream of unpacked bits.

Constructor Specific Documentation:

Make a CPFSK block.

Parameters:
- **k** – modulation index
- **ampl** – output amplitude
- **samples_per_sym** – number of output samples per input bit

cpfsk_bc_sptr.**active_thread_priority**(*cpfsk_bc_sptr self*) → int

cpfsk_bc_sptr.**amplitude**(*cpfsk_bc_sptr self*) → float

cpfsk_bc_sptr.**declare_sample_delay**(*cpfsk_bc_sptr self*, *int which*, *int delay*)

declare_sample_delay(cpfsk_bc_sptr self, unsigned int delay)

cpfsk_bc_sptr.**freq**(*cpfsk_bc_sptr self*) → float

cpfsk_bc_sptr.**message_subscribers**(*cpfsk_bc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

cpfsk_bc_sptr.**min_noutput_items**(*cpfsk_bc_sptr self*) → int

cpfsk_bc_sptr.**pc_input_buffers_full_avg**(*cpfsk_bc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(cpfsk_bc_sptr self) -> pmt_vector_float

cpfsk_bc_sptr.**pc_noutput_items_avg**(*cpfsk_bc_sptr self*) → float

cpfsk_bc_sptr.**pc_nproduced_avg**(*cpfsk_bc_sptr self*) → float

cpfsk_bc_sptr.**pc_output_buffers_full_avg**(*cpfsk_bc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(cpfsk_bc_sptr self) -> pmt_vector_float

cpfsk_bc_sptr.**pc_throughput_avg**(*cpfsk_bc_sptr self*) → float

cpfsk_bc_sptr.**pc_work_time_avg**(*cpfsk_bc_sptr self*) → float

cpfsk_bc_sptr.**pc_work_time_total**(*cpfsk_bc_sptr self*) → float

cpfsk_bc_sptr.**phase**(*cpfsk_bc_sptr self*) → float

cpfsk_bc_sptr.**sample_delay**(*cpfsk_bc_sptr self*, *int which*) → unsigned int

cpfsk_bc_sptr.**set_amplitude**(*cpfsk_bc_sptr self*, *float amplitude*)

cpfsk_bc_sptr.**set_min_noutput_items**(*cpfsk_bc_sptr self*, *int m*)

cpfsk_bc_sptr.**set_thread_priority**(*cpfsk_bc_sptr self*, *int priority*) → int

cpfsk_bc_sptr.**thread_priority**(*cpfsk_bc_sptr self*) → int

gnuradio.analog.**ctcss_squelch_ff**(*int rate*, *float freq*, *float level*, *int len*, *int ramp*, *bool gate*) → ctcss_squelch_ff_sptr

gate or zero output if CTCSS tone not present

Constructor Specific Documentation:

Make CTCSS tone squelch block.

Parameters:
- **rate** – gain of the internal frequency filters.
- **freq** – frequency value to use as the squelch tone.
- **level** – threshold level for the squelch tone.
- **len** – length of the frequency filters.
- **ramp** – sets response characteristic.
- **gate** – if true, no output if no squelch tone. if false, output 0's if no squelch tone.

ctcss_squelch_ff_sptr.**active_thread_priority**(*ctcss_squelch_ff_sptr self*) → int

ctcss_squelch_ff_sptr.**declare_sample_delay**(*ctcss_squelch_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(ctcss_squelch_ff_sptr self, unsigned int delay)

ctcss_squelch_ff_sptr.**frequency**(*ctcss_squelch_ff_sptr self*) → float

ctcss_squelch_ff_sptr.**gate**(*ctcss_squelch_ff_sptr self*) → bool

ctcss_squelch_ff_sptr.**len**(*ctcss_squelch_ff_sptr self*) → int

ctcss_squelch_ff_sptr.**level**(*ctcss_squelch_ff_sptr self*) → float

ctcss_squelch_ff_sptr.**message_subscribers**(*ctcss_squelch_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

ctcss_squelch_ff_sptr.**min_noutput_items**(*ctcss_squelch_ff_sptr self*) → int

ctcss_squelch_ff_sptr.**pc_input_buffers_full_avg**(*ctcss_squelch_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(ctcss_squelch_ff_sptr self) -> pmt_vector_float

ctcss_squelch_ff_sptr.**pc_noutput_items_avg**(*ctcss_squelch_ff_sptr self*) → float

ctcss_squelch_ff_sptr.**pc_nproduced_avg**(*ctcss_squelch_ff_sptr self*) → float

ctcss_squelch_ff_sptr.**pc_output_buffers_full_avg**(*ctcss_squelch_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(ctcss_squelch_ff_sptr self) -> pmt_vector_float

ctcss_squelch_ff_sptr.**pc_throughput_avg**(*ctcss_squelch_ff_sptr self*) → float

ctcss_squelch_ff_sptr.**pc_work_time_avg**(*ctcss_squelch_ff_sptr self*) → float

ctcss_squelch_ff_sptr.**pc_work_time_total**(*ctcss_squelch_ff_sptr self*) → float

ctcss_squelch_ff_sptr.**ramp**(*ctcss_squelch_ff_sptr self*) → int

ctcss_squelch_ff_sptr.**sample_delay**(*ctcss_squelch_ff_sptr self*, *int which*) → unsigned int

ctcss_squelch_ff_sptr.**set_frequency**(*ctcss_squelch_ff_sptr self*, *float frequency*)

ctcss_squelch_ff_sptr.**set_gate**(*ctcss_squelch_ff_sptr self*, *bool gate*)

ctcss_squelch_ff_sptr.**set_level**(*ctcss_squelch_ff_sptr self*, *float level*)

ctcss_squelch_ff_sptr.**set_min_noutput_items**(*ctcss_squelch_ff_sptr self*, *int m*)

ctcss_squelch_ff_sptr.**set_ramp**(*ctcss_squelch_ff_sptr self*, *int ramp*)

ctcss_squelch_ff_sptr.**set_thread_priority**(*ctcss_squelch_ff_sptr self*, *int priority*) → int

ctcss_squelch_ff_sptr.**squelch_range**(*ctcss_squelch_ff_sptr self*) → pmt_vector_float

ctcss_squelch_ff_sptr.**thread_priority**(*ctcss_squelch_ff_sptr self*) → int

ctcss_squelch_ff_sptr.**unmuted**(*ctcss_squelch_ff_sptr self*) → bool

gnuradio.analog.**dpll_bb**(*float period*, *float gain*) → dpll_bb_sptr

Detect the peak of a signal.

If a peak is detected, this block outputs a 1, or it outputs 0's.

Constructor Specific Documentation:

Parameters:
- **period** –
- **gain** –

dpll_bb_sptr.**active_thread_priority**(*dpll_bb_sptr self*) → int

dpll_bb_sptr.**decision_threshold**(*dpll_bb_sptr self*) → float

dpll_bb_sptr.**declare_sample_delay**(*dpll_bb_sptr self*, *int which*, *int delay*)

declare_sample_delay(dpll_bb_sptr self, unsigned int delay)

dpll_bb_sptr.**freq**(*dpll_bb_sptr self*) → float

dpll_bb_sptr.**gain**(*dpll_bb_sptr self*) → float

dpll_bb_sptr.**message_subscribers**(*dpll_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

dpll_bb_sptr.**min_noutput_items**(*dpll_bb_sptr self*) → int

dpll_bb_sptr.**pc_input_buffers_full_avg**(*dpll_bb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(dpll_bb_sptr self) -> pmt_vector_float

dpll_bb_sptr.**pc_noutput_items_avg**(*dpll_bb_sptr self*) → float

dpll_bb_sptr.**pc_nproduced_avg**(*dpll_bb_sptr self*) → float

dpll_bb_sptr.**pc_output_buffers_full_avg**(*dpll_bb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(dpll_bb_sptr self) -> pmt_vector_float

dpll_bb_sptr.**pc_throughput_avg**(*dpll_bb_sptr self*) → float

dpll_bb_sptr.**pc_work_time_avg**(*dpll_bb_sptr self*) → float

dpll_bb_sptr.**pc_work_time_total**(*dpll_bb_sptr self*) → float

dpll_bb_sptr.**phase**(*dpll_bb_sptr self*) → float

dpll_bb_sptr.**sample_delay**(*dpll_bb_sptr self*, *int which*) → unsigned int

dpll_bb_sptr.**set_decision_threshold**(*dpll_bb_sptr self*, *float thresh*)

dpll_bb_sptr.**set_gain**(*dpll_bb_sptr self*, *float gain*)

dpll_bb_sptr.**set_min_noutput_items**(*dpll_bb_sptr self*, *int m*)

dpll_bb_sptr.**set_thread_priority**(*dpll_bb_sptr self*, *int priority*) → int

dpll_bb_sptr.**thread_priority**(*dpll_bb_sptr self*) → int

gnuradio.analog.**fastnoise_source_c**(*gr::analog::noise_type_t type*, *float ampl*, *long seed=0*, *long samples=1024*) → fastnoise_source_c_sptr

Random number source.

Generate random values from different distributions. Currently, only Gaussian and uniform are enabled.

Constructor Specific Documentation:

Make a fast noise source.

Parameters:
- **type** – the random distribution to use (see gnuradio/analog/noise_type.h)
- **ampl** – the standard deviation of a 1-d noise process. If this is the complex source, this parameter is split among the real and imaginary parts:
- **seed** – seed for random generators. Note that for uniform and Gaussian distributions, this should be a negative number.
- **samples** – Number of samples to pre-generate

fastnoise_source_c_sptr.**active_thread_priority**(*fastnoise_source_c_sptr self*) → int

fastnoise_source_c_sptr.**amplitude**(*fastnoise_source_c_sptr self*) → float

fastnoise_source_c_sptr.**declare_sample_delay**(*fastnoise_source_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(fastnoise_source_c_sptr self, unsigned int delay)

fastnoise_source_c_sptr.**message_subscribers**(*fastnoise_source_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

fastnoise_source_c_sptr.**min_noutput_items**(*fastnoise_source_c_sptr self*) → int

fastnoise_source_c_sptr.**pc_input_buffers_full_avg**(*fastnoise_source_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(fastnoise_source_c_sptr self) -> pmt_vector_float

fastnoise_source_c_sptr.**pc_noutput_items_avg**(*fastnoise_source_c_sptr self*) → float

fastnoise_source_c_sptr.**pc_nproduced_avg**(*fastnoise_source_c_sptr self*) → float

fastnoise_source_c_sptr.**pc_output_buffers_full_avg**(*fastnoise_source_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(fastnoise_source_c_sptr self) -> pmt_vector_float

fastnoise_source_c_sptr.**pc_throughput_avg**(*fastnoise_source_c_sptr self*) → float

fastnoise_source_c_sptr.**pc_work_time_avg**(*fastnoise_source_c_sptr self*) → float

fastnoise_source_c_sptr.**pc_work_time_total**(*fastnoise_source_c_sptr self*) → float

fastnoise_source_c_sptr.**sample**(*fastnoise_source_c_sptr self*) → gr_complex

fastnoise_source_c_sptr.**sample_delay**(*fastnoise_source_c_sptr self*, *int which*) → unsigned int

fastnoise_source_c_sptr.**sample_unbiased**(*fastnoise_source_c_sptr self*) → gr_complex

fastnoise_source_c_sptr.**samples**(*fastnoise_source_c_sptr self*) → pmt_vector_cfloat

fastnoise_source_c_sptr.**set_amplitude**(*fastnoise_source_c_sptr self*, *float ampl*)

Set the standard deviation (amplitude) of the 1-d noise process.

fastnoise_source_c_sptr.**set_min_noutput_items**(*fastnoise_source_c_sptr self*, *int m*)

fastnoise_source_c_sptr.**set_thread_priority**(*fastnoise_source_c_sptr self*, *int priority*) → int

fastnoise_source_c_sptr.**set_type**(*fastnoise_source_c_sptr self*, *gr::analog::noise_type_t type*)

Set the noise type. Nominally from the gr::analog::noise_type_t selections, but only GR_GAUSSIAN and GR_UNIFORM are currently available.

fastnoise_source_c_sptr.**thread_priority**(*fastnoise_source_c_sptr self*) → int

gnuradio.analog.**fastnoise_source_f**(*gr::analog::noise_type_t type*, *float ampl*, *long seed=0*, *long samples=1024*) → fastnoise_source_f_sptr

Random number source.

Generate random values from different distributions. Currently, only Gaussian and uniform are enabled.

Constructor Specific Documentation:

Make a fast noise source.

Parameters:

- **type** – the random distribution to use (see gnuradio/analog/noise_type.h)
- **ampl** – the standard deviation of a 1-d noise process. If this is the complex source, this parameter is split among the real and imaginary parts:
- **seed** – seed for random generators. Note that for uniform and Gaussian distributions, this should be a negative number.
- **samples** – Number of samples to pre-generate

fastnoise_source_f_sptr.**active_thread_priority**(*fastnoise_source_f_sptr self*) → int

fastnoise_source_f_sptr.**amplitude**(*fastnoise_source_f_sptr self*) → float

fastnoise_source_f_sptr.**declare_sample_delay**(*fastnoise_source_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(fastnoise_source_f_sptr self, unsigned int delay)

fastnoise_source_f_sptr.**message_subscribers**(*fastnoise_source_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

fastnoise_source_f_sptr.**min_noutput_items**(*fastnoise_source_f_sptr self*) → int

fastnoise_source_f_sptr.**pc_input_buffers_full_avg**(*fastnoise_source_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(fastnoise_source_f_sptr self) -> pmt_vector_float

fastnoise_source_f_sptr.**pc_noutput_items_avg**(*fastnoise_source_f_sptr self*) → float

fastnoise_source_f_sptr.**pc_nproduced_avg**(*fastnoise_source_f_sptr self*) → float

fastnoise_source_f_sptr.**pc_output_buffers_full_avg**(*fastnoise_source_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(fastnoise_source_f_sptr self) -> pmt_vector_float

fastnoise_source_f_sptr.**pc_throughput_avg**(*fastnoise_source_f_sptr self*) → float

fastnoise_source_f_sptr.**pc_work_time_avg**(*fastnoise_source_f_sptr self*) → float

fastnoise_source_f_sptr.**pc_work_time_total**(*fastnoise_source_f_sptr self*) → float

fastnoise_source_f_sptr.**sample**(*fastnoise_source_f_sptr self*) → float

fastnoise_source_f_sptr.**sample_delay**(*fastnoise_source_f_sptr self*, *int which*) → unsigned int

fastnoise_source_f_sptr.**sample_unbiased**(*fastnoise_source_f_sptr self*) → float

fastnoise_source_f_sptr.**samples**(*fastnoise_source_f_sptr self*) → pmt_vector_float

fastnoise_source_f_sptr.**set_amplitude**(*fastnoise_source_f_sptr self*, *float ampl*)

Set the standard deviation (amplitude) of the 1-d noise process.

fastnoise_source_f_sptr.**set_min_noutput_items**(*fastnoise_source_f_sptr self*, *int m*)

fastnoise_source_f_sptr.**set_thread_priority**(*fastnoise_source_f_sptr self*, *int priority*) → int

fastnoise_source_f_sptr.**set_type**(*fastnoise_source_f_sptr self*, *gr::analog::noise_type_t type*)

Set the noise type. Nominally from the gr::analog::noise_type_t selections, but only GR_GAUSSIAN and GR_UNIFORM are currently available.

fastnoise_source_f_sptr.**thread_priority**(*fastnoise_source_f_sptr self*) → int

gnuradio.analog.**fastnoise_source_i**(*gr::analog::noise_type_t type*, *float ampl*, *long seed=0*, *long samples=1024*) → fastnoise_source_i_sptr

Random number source.

Generate random values from different distributions. Currently, only Gaussian and uniform are enabled.

Constructor Specific Documentation:

Make a fast noise source.

Parameters:

- **type** – the random distribution to use (see gnuradio/analog/noise_type.h)
- **ampl** – the standard deviation of a 1-d noise process. If this is the complex source, this parameter is split among the real and imaginary parts:
- **seed** – seed for random generators. Note that for uniform and Gaussian distributions, this should be a negative number.
- **samples** – Number of samples to pre-generate

fastnoise_source_i_sptr.**active_thread_priority**(*fastnoise_source_i_sptr self*) → int

fastnoise_source_i_sptr.**amplitude**(*fastnoise_source_i_sptr self*) → float

fastnoise_source_i_sptr.**declare_sample_delay**(*fastnoise_source_i_sptr self*, *int which*, *int delay*)

declare_sample_delay(fastnoise_source_i_sptr self, unsigned int delay)

fastnoise_source_i_sptr.**message_subscribers**(*fastnoise_source_i_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

fastnoise_source_i_sptr.**min_noutput_items**(*fastnoise_source_i_sptr self*) → int

fastnoise_source_i_sptr.**pc_input_buffers_full_avg**(*fastnoise_source_i_sptr self*, *int which*) → float

pc_input_buffers_full_avg(fastnoise_source_i_sptr self) -> pmt_vector_float

fastnoise_source_i_sptr.**pc_noutput_items_avg**(*fastnoise_source_i_sptr self*) → float

fastnoise_source_i_sptr.**pc_nproduced_avg**(*fastnoise_source_i_sptr self*) → float

fastnoise_source_i_sptr.**pc_output_buffers_full_avg**(*fastnoise_source_i_sptr self*, *int which*) → float

pc_output_buffers_full_avg(fastnoise_source_i_sptr self) -> pmt_vector_float

fastnoise_source_i_sptr.**pc_throughput_avg**(*fastnoise_source_i_sptr self*) → float

fastnoise_source_i_sptr.**pc_work_time_avg**(*fastnoise_source_i_sptr self*) → float

fastnoise_source_i_sptr.**pc_work_time_total**(*fastnoise_source_i_sptr self*) → float

fastnoise_source_i_sptr.**sample**(*fastnoise_source_i_sptr self*) → int

fastnoise_source_i_sptr.**sample_delay**(*fastnoise_source_i_sptr self*, *int which*) → unsigned int

fastnoise_source_i_sptr.**sample_unbiased**(*fastnoise_source_i_sptr self*) → int

fastnoise_source_i_sptr.**samples**(*fastnoise_source_i_sptr self*) → std::vector< int,std::allocator< int > > const &

fastnoise_source_i_sptr.**set_amplitude**(*fastnoise_source_i_sptr self*, *float ampl*)

Set the standard deviation (amplitude) of the 1-d noise process.

fastnoise_source_i_sptr.**set_min_noutput_items**(*fastnoise_source_i_sptr self*, *int m*)

fastnoise_source_i_sptr.**set_thread_priority**(*fastnoise_source_i_sptr self*, *int priority*) → int

fastnoise_source_i_sptr.**set_type**(*fastnoise_source_i_sptr self*, *gr::analog::noise_type_t type*)

Set the noise type. Nominally from the gr::analog::noise_type_t selections, but only GR_GAUSSIAN and GR_UNIFORM are currently available.

fastnoise_source_i_sptr.**thread_priority**(*fastnoise_source_i_sptr self*) → int

gnuradio.analog.**fastnoise_source_s**(*gr::analog::noise_type_t type*, *float ampl*, *long seed=0*, *long samples=1024*) → fastnoise_source_s_sptr

Random number source.

Generate random values from different distributions. Currently, only Gaussian and uniform are enabled.

Constructor Specific Documentation:

Make a fast noise source.

Parameters:
- **type** – the random distribution to use (see gnuradio/analog/noise_type.h)
- **ampl** – the standard deviation of a 1-d noise process. If this is the complex source, this parameter is split among the real and imaginary parts:
- **seed** – seed for random generators. Note that for uniform and Gaussian distributions, this should be a negative number.
- **samples** – Number of samples to pre-generate

fastnoise_source_s_sptr.**active_thread_priority**(*fastnoise_source_s_sptr self*) → int

fastnoise_source_s_sptr.**amplitude**(*fastnoise_source_s_sptr self*) → float

fastnoise_source_s_sptr.**declare_sample_delay**(*fastnoise_source_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(fastnoise_source_s_sptr self, unsigned int delay)

fastnoise_source_s_sptr.**message_subscribers**(*fastnoise_source_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

fastnoise_source_s_sptr.**min_noutput_items**(*fastnoise_source_s_sptr self*) → int

fastnoise_source_s_sptr.**pc_input_buffers_full_avg**(*fastnoise_source_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(fastnoise_source_s_sptr self) -> pmt_vector_float

fastnoise_source_s_sptr.**pc_noutput_items_avg**(*fastnoise_source_s_sptr self*) → float

fastnoise_source_s_sptr.**pc_nproduced_avg**(*fastnoise_source_s_sptr self*) → float

fastnoise_source_s_sptr.**pc_output_buffers_full_avg**(*fastnoise_source_s_sptr self*, *int which*) → float

pc_output_buffers_full_avg(fastnoise_source_s_sptr self) -> pmt_vector_float

fastnoise_source_s_sptr.**pc_throughput_avg**(*fastnoise_source_s_sptr self*) → float

fastnoise_source_s_sptr.**pc_work_time_avg**(*fastnoise_source_s_sptr self*) → float

fastnoise_source_s_sptr.**pc_work_time_total**(*fastnoise_source_s_sptr self*) → float

fastnoise_source_s_sptr.**sample**(*fastnoise_source_s_sptr self*) → short

fastnoise_source_s_sptr.**sample_delay**(*fastnoise_source_s_sptr self*, *int which*) → unsigned int

fastnoise_source_s_sptr.**sample_unbiased**(*fastnoise_source_s_sptr self*) → short

fastnoise_source_s_sptr.**samples**(*fastnoise_source_s_sptr self*) → std::vector< short,std::allocator< short > > const &

fastnoise_source_s_sptr.**set_amplitude**(*fastnoise_source_s_sptr self*, *float ampl*)

Set the standard deviation (amplitude) of the 1-d noise process.

fastnoise_source_s_sptr.**set_min_noutput_items**(*fastnoise_source_s_sptr self*, *int m*)

fastnoise_source_s_sptr.**set_thread_priority**(*fastnoise_source_s_sptr self*, *int priority*) → int

fastnoise_source_s_sptr.**set_type**(*fastnoise_source_s_sptr self*, *gr::analog::noise_type_t type*)

Set the noise type. Nominally from the gr::analog::noise_type_t selections, but only GR_GAUSSIAN and GR_UNIFORM are currently available.

fastnoise_source_s_sptr.**thread_priority**(*fastnoise_source_s_sptr self*) → int

gnuradio.analog.**feedforward_agc_cc**(*int nsamples*, *float reference*) → feedforward_agc_cc_sptr

Non-causal AGC which computes required gain based on max absolute value over nsamples.

Constructor Specific Documentation:

Build a complex valued feed-forward AGC loop block.

Parameters:
- **nsamples** – number of samples to look ahead.
- **reference** – reference value to adjust signal power to.

feedforward_agc_cc_sptr.**active_thread_priority**(*feedforward_agc_cc_sptr self*) → int

feedforward_agc_cc_sptr.**declare_sample_delay**(*feedforward_agc_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(feedforward_agc_cc_sptr self, unsigned int delay)

feedforward_agc_cc_sptr.**message_subscribers**(*feedforward_agc_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

feedforward_agc_cc_sptr.**min_noutput_items**(*feedforward_agc_cc_sptr self*) → int

feedforward_agc_cc_sptr.**pc_input_buffers_full_avg**(*feedforward_agc_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(feedforward_agc_cc_sptr self) -> pmt_vector_float

feedforward_agc_cc_sptr.**pc_noutput_items_avg**(*feedforward_agc_cc_sptr self*) → float

feedforward_agc_cc_sptr.**pc_nproduced_avg**(*feedforward_agc_cc_sptr self*) → float

feedforward_agc_cc_sptr.**pc_output_buffers_full_avg**(*feedforward_agc_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(feedforward_agc_cc_sptr self) -> pmt_vector_float

feedforward_agc_cc_sptr.**pc_throughput_avg**(*feedforward_agc_cc_sptr self*) → float

feedforward_agc_cc_sptr.**pc_work_time_avg**(*feedforward_agc_cc_sptr self*) → float

feedforward_agc_cc_sptr.**pc_work_time_total**(*feedforward_agc_cc_sptr self*) → float

feedforward_agc_cc_sptr.**sample_delay**(*feedforward_agc_cc_sptr self*, *int which*) → unsigned int

feedforward_agc_cc_sptr.**set_min_noutput_items**(*feedforward_agc_cc_sptr self*, *int m*)

feedforward_agc_cc_sptr.**set_thread_priority**(*feedforward_agc_cc_sptr self*, *int priority*) → int

feedforward_agc_cc_sptr.**thread_priority**(*feedforward_agc_cc_sptr self*) → int

gnuradio.analog.**fmdet_cf**(*float samplerate*, *float freq_low*, *float freq_high*, *float scl*) → fmdet_cf_sptr

Implements an IQ slope detector.

input: stream of complex; output: stream of floats

This implements a limiting slope detector. The limiter is in the normalization by the magnitude of the sample

Constructor Specific Documentation:

Make FM detector block.

Parameters:
- **samplerate** – sample rate of signal (is not used; to be removed)
- **freq_low** – lowest frequency of signal (Hz)
- **freq_high** – highest frequency of signal (Hz)
- **scl** – scale factor

fmdet_cf_sptr.**active_thread_priority**(*fmdet_cf_sptr self*) → int

fmdet_cf_sptr.**bias**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**declare_sample_delay**(*fmdet_cf_sptr self*, *int which*, *int delay*)

declare_sample_delay(fmdet_cf_sptr self, unsigned int delay)

fmdet_cf_sptr.**freq**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**freq_high**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**freq_low**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**message_subscribers**(*fmdet_cf_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

fmdet_cf_sptr.**min_noutput_items**(*fmdet_cf_sptr self*) → int

fmdet_cf_sptr.**pc_input_buffers_full_avg**(*fmdet_cf_sptr self*, *int which*) → float

pc_input_buffers_full_avg(fmdet_cf_sptr self) -> pmt_vector_float

fmdet_cf_sptr.**pc_noutput_items_avg**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**pc_nproduced_avg**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**pc_output_buffers_full_avg**(*fmdet_cf_sptr self*, *int which*) → float

pc_output_buffers_full_avg(fmdet_cf_sptr self) -> pmt_vector_float

fmdet_cf_sptr.**pc_throughput_avg**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**pc_work_time_avg**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**pc_work_time_total**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**sample_delay**(*fmdet_cf_sptr self*, *int which*) → unsigned int

fmdet_cf_sptr.**scale**(*fmdet_cf_sptr self*) → float

fmdet_cf_sptr.**set_freq_range**(*fmdet_cf_sptr self*, *float freq_low*, *float freq_high*)

fmdet_cf_sptr.**set_min_noutput_items**(*fmdet_cf_sptr self*, *int m*)

fmdet_cf_sptr.**set_scale**(*fmdet_cf_sptr self*, *float scl*)

fmdet_cf_sptr.**set_thread_priority**(*fmdet_cf_sptr self*, *int priority*) → int

fmdet_cf_sptr.**thread_priority**(*fmdet_cf_sptr self*) → int

gnuradio.analog.**frequency_modulator_fc**(*float sensitivity*) → frequency_modulator_fc_sptr

Frequency modulator block.

float input; complex baseband output

Takes a real, baseband signal (x_m[n]) and output a frequency modulated signal (y[n]) according to:

Where x[n] is the input sample at time n and is the frequency deviation. Common values for are 5 kHz for narrowband FM channels such as for voice systems and 75 KHz for wideband FM, like audio broadcast FM stations.

In this block, the input argument is , not the frequency deviation. The sensitivity specifies how much the phase changes based on the new input sample. Given a maximum deviation, , and sample rate , the sensitivity is defined as:

Constructor Specific Documentation:

Build a frequency modulator block.

Parameters: **sensitivity** – radians/sample = amplitude * sensitivity

frequency_modulator_fc_sptr.**active_thread_priority**(*frequency_modulator_fc_sptr self*) → int

frequency_modulator_fc_sptr.**declare_sample_delay**(*frequency_modulator_fc_sptr self*, *int which*, *int delay*)

declare_sample_delay(frequency_modulator_fc_sptr self, unsigned int delay)

frequency_modulator_fc_sptr.**message_subscribers**(*frequency_modulator_fc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

frequency_modulator_fc_sptr.**min_noutput_items**(*frequency_modulator_fc_sptr self*) → int

frequency_modulator_fc_sptr.**pc_input_buffers_full_avg**(*frequency_modulator_fc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(frequency_modulator_fc_sptr self) -> pmt_vector_float

frequency_modulator_fc_sptr.**pc_noutput_items_avg**(*frequency_modulator_fc_sptr self*) → float

frequency_modulator_fc_sptr.**pc_nproduced_avg**(*frequency_modulator_fc_sptr self*) → float

frequency_modulator_fc_sptr.**pc_output_buffers_full_avg**(*frequency_modulator_fc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(frequency_modulator_fc_sptr self) -> pmt_vector_float

frequency_modulator_fc_sptr.**pc_throughput_avg**(*frequency_modulator_fc_sptr self*) → float

frequency_modulator_fc_sptr.**pc_work_time_avg**(*frequency_modulator_fc_sptr self*) → float

frequency_modulator_fc_sptr.**pc_work_time_total**(*frequency_modulator_fc_sptr self*) → float

frequency_modulator_fc_sptr.**sample_delay**(*frequency_modulator_fc_sptr self*, *int which*) → unsigned int

frequency_modulator_fc_sptr.**sensitivity**(*frequency_modulator_fc_sptr self*) → float

frequency_modulator_fc_sptr.**set_min_noutput_items**(*frequency_modulator_fc_sptr self*, *int m*)

frequency_modulator_fc_sptr.**set_sensitivity**(*frequency_modulator_fc_sptr self*, *float sens*)

frequency_modulator_fc_sptr.**set_thread_priority**(*frequency_modulator_fc_sptr self*, *int priority*) → int

frequency_modulator_fc_sptr.**thread_priority**(*frequency_modulator_fc_sptr self*) → int

gnuradio.analog.**noise_source_c**(*gr::analog::noise_type_t type*, *float ampl*, *long seed=0*) → noise_source_c_sptr

Random number source.

Generate random values from different distributions. Currently, only Gaussian and uniform are enabled.

Constructor Specific Documentation:

Build a noise source

Parameters:
- **type** – the random distribution to use (see gnuradio/analog/noise_type.h)
- **ampl** – the standard deviation of a 1-d noise process. If this is the complex source, this parameter is split among the real and imaginary parts:
- **seed** – seed for random generators. Note that for uniform and Gaussian distributions, this should be a negative number.

noise_source_c_sptr.**active_thread_priority**(*noise_source_c_sptr self*) → int

noise_source_c_sptr.**amplitude**(*noise_source_c_sptr self*) → float

noise_source_c_sptr.**declare_sample_delay**(*noise_source_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(noise_source_c_sptr self, unsigned int delay)

noise_source_c_sptr.**message_subscribers**(*noise_source_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

noise_source_c_sptr.**min_noutput_items**(*noise_source_c_sptr self*) → int

noise_source_c_sptr.**pc_input_buffers_full_avg**(*noise_source_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(noise_source_c_sptr self) -> pmt_vector_float

noise_source_c_sptr.**pc_noutput_items_avg**(*noise_source_c_sptr self*) → float

noise_source_c_sptr.**pc_nproduced_avg**(*noise_source_c_sptr self*) → float

noise_source_c_sptr.**pc_output_buffers_full_avg**(*noise_source_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(noise_source_c_sptr self) -> pmt_vector_float

noise_source_c_sptr.**pc_throughput_avg**(*noise_source_c_sptr self*) → float

noise_source_c_sptr.**pc_work_time_avg**(*noise_source_c_sptr self*) → float

noise_source_c_sptr.**pc_work_time_total**(*noise_source_c_sptr self*) → float

noise_source_c_sptr.**sample_delay**(*noise_source_c_sptr self*, *int which*) → unsigned int

noise_source_c_sptr.**set_amplitude**(*noise_source_c_sptr self*, *float ampl*)

Set the standard deviation (amplitude) of the 1-d noise process.

noise_source_c_sptr.**set_min_noutput_items**(*noise_source_c_sptr self*, *int m*)

noise_source_c_sptr.**set_thread_priority**(*noise_source_c_sptr self*, *int priority*) → int

noise_source_c_sptr.**set_type**(*noise_source_c_sptr self*, *gr::analog::noise_type_t type*)

Set the noise type. Nominally from the gr::analog::noise_type_t selections, but only GR_GAUSSIAN and GR_UNIFORM are currently available.

noise_source_c_sptr.**thread_priority**(*noise_source_c_sptr self*) → int

gnuradio.analog.**noise_source_f**(*gr::analog::noise_type_t type*, *float ampl*, *long seed=0*) → noise_source_f_sptr

Random number source.

Generate random values from different distributions. Currently, only Gaussian and uniform are enabled.

Constructor Specific Documentation:

Build a noise source

Parameters:

- **type** – the random distribution to use (see gnuradio/analog/noise_type.h)
- **ampl** – the standard deviation of a 1-d noise process. If this is the complex source, this parameter is split among the real and imaginary parts:
- **seed** – seed for random generators. Note that for uniform and Gaussian distributions, this should be a negative number.

noise_source_f_sptr.**active_thread_priority**(*noise_source_f_sptr self*) → int

noise_source_f_sptr.**amplitude**(*noise_source_f_sptr self*) → float

noise_source_f_sptr.**declare_sample_delay**(*noise_source_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(noise_source_f_sptr self, unsigned int delay)

noise_source_f_sptr.**message_subscribers**(*noise_source_f_sptr self*, *swig_int_ptr*

which_port) → swig_int_ptr

noise_source_f_sptr.**min_noutput_items**(*noise_source_f_sptr self*) → int

noise_source_f_sptr.**pc_input_buffers_full_avg**(*noise_source_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(noise_source_f_sptr self) -> pmt_vector_float

noise_source_f_sptr.**pc_noutput_items_avg**(*noise_source_f_sptr self*) → float

noise_source_f_sptr.**pc_nproduced_avg**(*noise_source_f_sptr self*) → float

noise_source_f_sptr.**pc_output_buffers_full_avg**(*noise_source_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(noise_source_f_sptr self) -> pmt_vector_float

noise_source_f_sptr.**pc_throughput_avg**(*noise_source_f_sptr self*) → float

noise_source_f_sptr.**pc_work_time_avg**(*noise_source_f_sptr self*) → float

noise_source_f_sptr.**pc_work_time_total**(*noise_source_f_sptr self*) → float

noise_source_f_sptr.**sample_delay**(*noise_source_f_sptr self*, *int which*) → unsigned int

noise_source_f_sptr.**set_amplitude**(*noise_source_f_sptr self*, *float ampl*)

Set the standard deviation (amplitude) of the 1-d noise process.

noise_source_f_sptr.**set_min_noutput_items**(*noise_source_f_sptr self*, *int m*)

noise_source_f_sptr.**set_thread_priority**(*noise_source_f_sptr self*, *int priority*) → int

noise_source_f_sptr.**set_type**(*noise_source_f_sptr self*, *gr::analog::noise_type_t type*)

Set the noise type. Nominally from the gr::analog::noise_type_t selections, but only GR_GAUSSIAN and GR_UNIFORM are currently available.

noise_source_f_sptr.**thread_priority**(*noise_source_f_sptr self*) → int

gnuradio.analog.**noise_source_i**(*gr::analog::noise_type_t type*, *float ampl*, *long seed=0*) → noise_source_i_sptr

Random number source.

Generate random values from different distributions. Currently, only Gaussian and uniform are enabled.

Constructor Specific Documentation:

Build a noise source

Parameters:
- **type** – the random distribution to use (see gnuradio/analog/noise_type.h)
- **ampl** – the standard deviation of a 1-d noise process. If this is the complex source, this parameter is split among the real and imaginary parts:
- **seed** – seed for random generators. Note that for uniform and Gaussian distributions, this should be a negative number.

noise_source_i_sptr.**active_thread_priority**(*noise_source_i_sptr self*) → int

noise_source_i_sptr.**amplitude**(*noise_source_i_sptr self*) → float

noise_source_i_sptr.**declare_sample_delay**(*noise_source_i_sptr self*, *int which*, *int delay*)

declare_sample_delay(noise_source_i_sptr self, unsigned int delay)

noise_source_i_sptr.**message_subscribers**(*noise_source_i_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

noise_source_i_sptr.**min_noutput_items**(*noise_source_i_sptr self*) → int

noise_source_i_sptr.**pc_input_buffers_full_avg**(*noise_source_i_sptr self*, *int which*) → float

pc_input_buffers_full_avg(noise_source_i_sptr self) -> pmt_vector_float

noise_source_i_sptr.**pc_noutput_items_avg**(*noise_source_i_sptr self*) → float

noise_source_i_sptr.**pc_nproduced_avg**(*noise_source_i_sptr self*) → float

noise_source_i_sptr.**pc_output_buffers_full_avg**(*noise_source_i_sptr self*, *int which*) → float

pc_output_buffers_full_avg(noise_source_i_sptr self) -> pmt_vector_float

noise_source_i_sptr.**pc_throughput_avg**(*noise_source_i_sptr self*) → float

noise_source_i_sptr.**pc_work_time_avg**(*noise_source_i_sptr self*) → float

noise_source_i_sptr.**pc_work_time_total**(*noise_source_i_sptr self*) → float

noise_source_i_sptr.**sample_delay**(*noise_source_i_sptr self*, *int which*) → unsigned int

noise_source_i_sptr.**set_amplitude**(*noise_source_i_sptr self*, *float ampl*)

Set the standard deviation (amplitude) of the 1-d noise process.

noise_source_i_sptr.**set_min_noutput_items**(*noise_source_i_sptr self*, *int m*)

noise_source_i_sptr.**set_thread_priority**(*noise_source_i_sptr self*, *int priority*) → int

noise_source_i_sptr.**set_type**(*noise_source_i_sptr self*, *gr::analog::noise_type_t type*)

Set the noise type. Nominally from the gr::analog::noise_type_t selections, but only GR_GAUSSIAN and GR_UNIFORM are currently available.

noise_source_i_sptr.**thread_priority**(*noise_source_i_sptr self*) → int

gnuradio.analog.**noise_source_s**(*gr::analog::noise_type_t type*, *float ampl*, *long seed=0*) → noise_source_s_sptr

Random number source.

Generate random values from different distributions. Currently, only Gaussian and uniform are enabled.

Constructor Specific Documentation:

Build a noise source

Parameters:
- **type** – the random distribution to use (see gnuradio/analog/noise_type.h)
- **ampl** – the standard deviation of a 1-d noise process. If this is the complex source, this parameter is split among the real and imaginary parts:
- **seed** – seed for random generators. Note that for uniform and Gaussian distributions, this should be a negative number.

noise_source_s_sptr.**active_thread_priority**(*noise_source_s_sptr self*) → int

noise_source_s_sptr.**amplitude**(*noise_source_s_sptr self*) → float

noise_source_s_sptr.**declare_sample_delay**(*noise_source_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(noise_source_s_sptr self, unsigned int delay)

noise_source_s_sptr.**message_subscribers**(*noise_source_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

noise_source_s_sptr.**min_noutput_items**(*noise_source_s_sptr self*) → int

noise_source_s_sptr.**pc_input_buffers_full_avg**(*noise_source_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(noise_source_s_sptr self) -> pmt_vector_float

noise_source_s_sptr.**pc_noutput_items_avg**(*noise_source_s_sptr self*) → float

noise_source_s_sptr.**pc_nproduced_avg**(*noise_source_s_sptr self*) → float

noise_source_s_sptr.**pc_output_buffers_full_avg**(*noise_source_s_sptr self*, *int*

which) → float

pc_output_buffers_full_avg(noise_source_s_sptr self) -> pmt_vector_float

noise_source_s_sptr.**pc_throughput_avg**(*noise_source_s_sptr self*) → float

noise_source_s_sptr.**pc_work_time_avg**(*noise_source_s_sptr self*) → float

noise_source_s_sptr.**pc_work_time_total**(*noise_source_s_sptr self*) → float

noise_source_s_sptr.**sample_delay**(*noise_source_s_sptr self*, *int which*) → unsigned int

noise_source_s_sptr.**set_amplitude**(*noise_source_s_sptr self*, *float ampl*)

Set the standard deviation (amplitude) of the 1-d noise process.

noise_source_s_sptr.**set_min_noutput_items**(*noise_source_s_sptr self*, *int m*)

noise_source_s_sptr.**set_thread_priority**(*noise_source_s_sptr self*, *int priority*) → int

noise_source_s_sptr.**set_type**(*noise_source_s_sptr self*, *gr::analog::noise_type_t type*)

Set the noise type. Nominally from the gr::analog::noise_type_t selections, but only GR_GAUSSIAN and GR_UNIFORM are currently available.

noise_source_s_sptr.**thread_priority**(*noise_source_s_sptr self*) → int

gnuradio.analog.**phase_modulator_fc**(*double sensitivity*) → phase_modulator_fc_sptr

Phase modulator block.

output = complex(cos(in*sensitivity), sin(in*sensitivity))

Input stream 0: floats Output stream 0: complex

Constructor Specific Documentation:

Parameters: **sensitivity** –

phase_modulator_fc_sptr.**active_thread_priority**(*phase_modulator_fc_sptr self*) → int

phase_modulator_fc_sptr.**declare_sample_delay**(*phase_modulator_fc_sptr self*, *int which*, *int delay*)

declare_sample_delay(phase_modulator_fc_sptr self, unsigned int delay)

phase_modulator_fc_sptr.**message_subscribers**(*phase_modulator_fc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

phase_modulator_fc_sptr.**min_noutput_items**(*phase_modulator_fc_sptr self*) → int

phase_modulator_fc_sptr.**pc_input_buffers_full_avg**(*phase_modulator_fc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(phase_modulator_fc_sptr self) -> pmt_vector_float

phase_modulator_fc_sptr.**pc_noutput_items_avg**(*phase_modulator_fc_sptr self*) → float

phase_modulator_fc_sptr.**pc_nproduced_avg**(*phase_modulator_fc_sptr self*) → float

phase_modulator_fc_sptr.**pc_output_buffers_full_avg**(*phase_modulator_fc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(phase_modulator_fc_sptr self) -> pmt_vector_float

phase_modulator_fc_sptr.**pc_throughput_avg**(*phase_modulator_fc_sptr self*) → float

phase_modulator_fc_sptr.**pc_work_time_avg**(*phase_modulator_fc_sptr self*) → float

phase_modulator_fc_sptr.**pc_work_time_total**(*phase_modulator_fc_sptr self*) → float

phase_modulator_fc_sptr.**phase**(*phase_modulator_fc_sptr self*) → double

phase_modulator_fc_sptr.**sample_delay**(*phase_modulator_fc_sptr self*, *int which*) → unsigned int

phase_modulator_fc_sptr.**sensitivity**(*phase_modulator_fc_sptr self*) → double

phase_modulator_fc_sptr.**set_min_noutput_items**(*phase_modulator_fc_sptr self*, *int m*)

phase_modulator_fc_sptr.**set_phase**(*phase_modulator_fc_sptr self*, *double p*)

phase_modulator_fc_sptr.**set_sensitivity**(*phase_modulator_fc_sptr self*, *double s*)

phase_modulator_fc_sptr.**set_thread_priority**(*phase_modulator_fc_sptr self*, *int priority*) → int

phase_modulator_fc_sptr.**thread_priority**(*phase_modulator_fc_sptr self*) → int

gnuradio.analog.**pll_carriertracking_cc**(*float loop_bw*, *float max_freq*, *float min_freq*) → pll_carriertracking_cc_sptr

Implements a PLL which locks to the input frequency and outputs the input signal mixed with that carrier.

Input stream 0: complex Output stream 0: complex

This PLL locks onto a [possibly noisy] reference carrier on the input and outputs that signal, downconverted to DC

All settings max_freq and min_freq are in terms of radians per sample, NOT HERTZ. The loop bandwidth determines the lock range and should be set around pi/200 2pi/100.

Constructor Specific Documentation:

Parameters:
- **loop_bw** –
- **max_freq** –
- **min_freq** –

pll_carriertracking_cc_sptr.**active_thread_priority**(*pll_carriertracking_cc_sptr self*) → int

pll_carriertracking_cc_sptr.**advance_loop**(*pll_carriertracking_cc_sptr self*, *float error*)

pll_carriertracking_cc_sptr.**declare_sample_delay**(*pll_carriertracking_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(pll_carriertracking_cc_sptr self, unsigned int delay)

pll_carriertracking_cc_sptr.**frequency_limit**(*pll_carriertracking_cc_sptr self*)

pll_carriertracking_cc_sptr.**get_alpha**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**get_beta**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**get_damping_factor**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**get_frequency**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**get_loop_bandwidth**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**get_max_freq**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**get_min_freq**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**get_phase**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**lock_detector**(*pll_carriertracking_cc_sptr self*) → bool

pll_carriertracking_cc_sptr.**message_subscribers**(*pll_carriertracking_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pll_carriertracking_cc_sptr.**min_noutput_items**(*pll_carriertracking_cc_sptr self*) → int

pll_carriertracking_cc_sptr.**pc_input_buffers_full_avg**(*pll_carriertracking_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(pll_carriertracking_cc_sptr self) -> pmt_vector_float

pll_carriertracking_cc_sptr.**pc_noutput_items_avg**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**pc_nproduced_avg**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**pc_output_buffers_full_avg**(*pll_carriertracking_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(pll_carriertracking_cc_sptr self) -> pmt_vector_float

pll_carriertracking_cc_sptr.**pc_throughput_avg**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**pc_work_time_avg**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**pc_work_time_total**(*pll_carriertracking_cc_sptr self*) → float

pll_carriertracking_cc_sptr.**phase_wrap**(*pll_carriertracking_cc_sptr self*)

pll_carriertracking_cc_sptr.**sample_delay**(*pll_carriertracking_cc_sptr self*, *int which*) → unsigned int

pll_carriertracking_cc_sptr.**set_alpha**(*pll_carriertracking_cc_sptr self*, *float alpha*)

pll_carriertracking_cc_sptr.**set_beta**(*pll_carriertracking_cc_sptr self*, *float beta*)

pll_carriertracking_cc_sptr.**set_damping_factor**(*pll_carriertracking_cc_sptr self*, *float df*)

pll_carriertracking_cc_sptr.**set_frequency**(*pll_carriertracking_cc_sptr self*, *float freq*)

pll_carriertracking_cc_sptr.**set_lock_threshold**(*pll_carriertracking_cc_sptr self*, *float arg2*) → float

pll_carriertracking_cc_sptr.**set_loop_bandwidth**(*pll_carriertracking_cc_sptr self*, *float bw*)

pll_carriertracking_cc_sptr.**set_max_freq**(*pll_carriertracking_cc_sptr self*, *float freq*)

pll_carriertracking_cc_sptr.**set_min_freq**(*pll_carriertracking_cc_sptr self*, *float freq*)

pll_carriertracking_cc_sptr.**set_min_noutput_items**(*pll_carriertracking_cc_sptr self*, *int m*)

pll_carriertracking_cc_sptr.**set_phase**(*pll_carriertracking_cc_sptr self*, *float phase*)

pll_carriertracking_cc_sptr.**set_thread_priority**(*pll_carriertracking_cc_sptr self*, *int priority*) → int

pll_carriertracking_cc_sptr.**squelch_enable**(*pll_carriertracking_cc_sptr self*, *bool arg2*) → bool

pll_carriertracking_cc_sptr.**thread_priority**(*pll_carriertracking_cc_sptr self*) → int

pll_carriertracking_cc_sptr.**update_gains**(*pll_carriertracking_cc_sptr self*)

gnuradio.analog.**pll_freqdet_cf**(*float loop_bw*, *float max_freq*, *float min_freq*) → pll_freqdet_cf_sptr

Implements a PLL which locks to the input frequency and outputs an estimate of that frequency. Useful for FM Demod.

Input stream 0: complex Output stream 0: float

This PLL locks onto a [possibly noisy] reference carrier on the input and outputs an estimate of that frequency in radians per sample. All settings max_freq and min_freq are in terms of radians per sample, NOT HERTZ. The loop bandwidth determines the lock range and should be set around pi/200 2pi/100.

Constructor Specific Documentation:

Parameters:
- **loop_bw** –
- **max_freq** –
- **min_freq** –

pll_freqdet_cf_sptr.**active_thread_priority**(*pll_freqdet_cf_sptr self*) → int

pll_freqdet_cf_sptr.**advance_loop**(*pll_freqdet_cf_sptr self*, *float error*)

pll_freqdet_cf_sptr.**declare_sample_delay**(*pll_freqdet_cf_sptr self*, *int which*, *int delay*)

declare_sample_delay(pll_freqdet_cf_sptr self, unsigned int delay)

pll_freqdet_cf_sptr.**frequency_limit**(*pll_freqdet_cf_sptr self*)

pll_freqdet_cf_sptr.**get_alpha**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**get_beta**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**get_damping_factor**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**get_frequency**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**get_loop_bandwidth**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**get_max_freq**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**get_min_freq**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**get_phase**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**message_subscribers**(*pll_freqdet_cf_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pll_freqdet_cf_sptr.**min_noutput_items**(*pll_freqdet_cf_sptr self*) → int

pll_freqdet_cf_sptr.**pc_input_buffers_full_avg**(*pll_freqdet_cf_sptr self*, *int which*) → float

pc_input_buffers_full_avg(pll_freqdet_cf_sptr self) -> pmt_vector_float

pll_freqdet_cf_sptr.**pc_noutput_items_avg**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**pc_nproduced_avg**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**pc_output_buffers_full_avg**(*pll_freqdet_cf_sptr self*, *int which*) → float

pc_output_buffers_full_avg(pll_freqdet_cf_sptr self) -> pmt_vector_float

pll_freqdet_cf_sptr.**pc_throughput_avg**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**pc_work_time_avg**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**pc_work_time_total**(*pll_freqdet_cf_sptr self*) → float

pll_freqdet_cf_sptr.**phase_wrap**(*pll_freqdet_cf_sptr self*)

pll_freqdet_cf_sptr.**sample_delay**(*pll_freqdet_cf_sptr self*, *int which*) → unsigned int

pll_freqdet_cf_sptr.**set_alpha**(*pll_freqdet_cf_sptr self*, *float alpha*)

pll_freqdet_cf_sptr.**set_beta**(*pll_freqdet_cf_sptr self*, *float beta*)

pll_freqdet_cf_sptr.**set_damping_factor**(*pll_freqdet_cf_sptr self*, *float df*)

pll_freqdet_cf_sptr.**set_frequency**(*pll_freqdet_cf_sptr self*, *float freq*)

pll_freqdet_cf_sptr.**set_loop_bandwidth**(*pll_freqdet_cf_sptr self*, *float bw*)

pll_freqdet_cf_sptr.**set_max_freq**(*pll_freqdet_cf_sptr self*, *float freq*)

pll_freqdet_cf_sptr.**set_min_freq**(*pll_freqdet_cf_sptr self*, *float freq*)

pll_freqdet_cf_sptr.**set_min_noutput_items**(*pll_freqdet_cf_sptr self*, *int m*)

pll_freqdet_cf_sptr.**set_phase**(*pll_freqdet_cf_sptr self*, *float phase*)

pll_freqdet_cf_sptr.**set_thread_priority**(*pll_freqdet_cf_sptr self*, *int priority*) → int

pll_freqdet_cf_sptr.**thread_priority**(*pll_freqdet_cf_sptr self*) → int

pll_freqdet_cf_sptr.**update_gains**(*pll_freqdet_cf_sptr self*)

gnuradio.analog.**pll_refout_cc**(*float loop_bw*, *float max_freq*, *float min_freq*) → pll_refout_cc_sptr

Implements a PLL which locks to the input frequency and outputs a carrier.

Input stream 0: complex Output stream 0: complex

This PLL locks onto a [possibly noisy] reference carrier on the input and outputs a clean version which is phase and frequency aligned to it.

All settings max_freq and min_freq are in terms of radians per sample, NOT HERTZ. The loop bandwidth determines the lock range and should be set around pi/200 2pi/100.

Constructor Specific Documentation:

Parameters:
- **loop_bw** –
- **max_freq** –
- **min_freq** –

pll_refout_cc_sptr.**active_thread_priority**(*pll_refout_cc_sptr self*) → int

pll_refout_cc_sptr.**advance_loop**(*pll_refout_cc_sptr self*, *float error*)

pll_refout_cc_sptr.**declare_sample_delay**(*pll_refout_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(pll_refout_cc_sptr self, unsigned int delay)

pll_refout_cc_sptr.**frequency_limit**(*pll_refout_cc_sptr self*)

pll_refout_cc_sptr.**get_alpha**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**get_beta**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**get_damping_factor**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**get_frequency**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**get_loop_bandwidth**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**get_max_freq**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**get_min_freq**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**get_phase**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**message_subscribers**(*pll_refout_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pll_refout_cc_sptr.**min_noutput_items**(*pll_refout_cc_sptr self*) → int

pll_refout_cc_sptr.**pc_input_buffers_full_avg**(*pll_refout_cc_sptr self*, *int which*) →

float

pc_input_buffers_full_avg(pll_refout_cc_sptr self) -> pmt_vector_float

pll_refout_cc_sptr.**pc_noutput_items_avg**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**pc_nproduced_avg**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**pc_output_buffers_full_avg**(*pll_refout_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(pll_refout_cc_sptr self) -> pmt_vector_float

pll_refout_cc_sptr.**pc_throughput_avg**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**pc_work_time_avg**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**pc_work_time_total**(*pll_refout_cc_sptr self*) → float

pll_refout_cc_sptr.**phase_wrap**(*pll_refout_cc_sptr self*)

pll_refout_cc_sptr.**sample_delay**(*pll_refout_cc_sptr self*, *int which*) → unsigned int

pll_refout_cc_sptr.**set_alpha**(*pll_refout_cc_sptr self*, *float alpha*)

pll_refout_cc_sptr.**set_beta**(*pll_refout_cc_sptr self*, *float beta*)

pll_refout_cc_sptr.**set_damping_factor**(*pll_refout_cc_sptr self*, *float df*)

pll_refout_cc_sptr.**set_frequency**(*pll_refout_cc_sptr self*, *float freq*)

pll_refout_cc_sptr.**set_loop_bandwidth**(*pll_refout_cc_sptr self*, *float bw*)

pll_refout_cc_sptr.**set_max_freq**(*pll_refout_cc_sptr self*, *float freq*)

pll_refout_cc_sptr.**set_min_freq**(*pll_refout_cc_sptr self*, *float freq*)

pll_refout_cc_sptr.**set_min_noutput_items**(*pll_refout_cc_sptr self*, *int m*)

pll_refout_cc_sptr.**set_phase**(*pll_refout_cc_sptr self*, *float phase*)

pll_refout_cc_sptr.**set_thread_priority**(*pll_refout_cc_sptr self*, *int priority*) → int

pll_refout_cc_sptr.**thread_priority**(*pll_refout_cc_sptr self*) → int

pll_refout_cc_sptr.**update_gains**(*pll_refout_cc_sptr self*)

gnuradio.analog.**probe_avg_mag_sqrd_c**(*double threshold_db*, *double alpha=0.0001*) → probe_avg_mag_sqrd_c_sptr

compute avg magnitude squared.

Input stream 0: complex

Compute a running average of the magnitude squared of the the input. The level and indication as to whether the level exceeds threshold can be retrieved with the level and unmuted accessors.

Constructor Specific Documentation:

Make a complex sink that computes avg magnitude squared.

Parameters:
- **threshold_db** – Threshold for muting.
- **alpha** – Gain parameter for the running average filter.

probe_avg_mag_sqrd_c_sptr.**active_thread_priority**(*probe_avg_mag_sqrd_c_sptr self*) → int

probe_avg_mag_sqrd_c_sptr.**declare_sample_delay**(*probe_avg_mag_sqrd_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_avg_mag_sqrd_c_sptr self, unsigned int delay)

probe_avg_mag_sqrd_c_sptr.**level**(*probe_avg_mag_sqrd_c_sptr self*) → double

probe_avg_mag_sqrd_c_sptr.**message_subscribers**(*probe_avg_mag_sqrd_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_avg_mag_sqrd_c_sptr.**min_noutput_items**(*probe_avg_mag_sqrd_c_sptr self*) → int

probe_avg_mag_sqrd_c_sptr.**pc_input_buffers_full_avg**(*probe_avg_mag_sqrd_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_avg_mag_sqrd_c_sptr self) -> pmt_vector_float

probe_avg_mag_sqrd_c_sptr.**pc_noutput_items_avg**(*probe_avg_mag_sqrd_c_sptr self*) → float

probe_avg_mag_sqrd_c_sptr.**pc_nproduced_avg**(*probe_avg_mag_sqrd_c_sptr self*) → float

probe_avg_mag_sqrd_c_sptr.**pc_output_buffers_full_avg**(*probe_avg_mag_sqrd_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_avg_mag_sqrd_c_sptr self) -> pmt_vector_float

probe_avg_mag_sqrd_c_sptr.**pc_throughput_avg**(*probe_avg_mag_sqrd_c_sptr self*) → float

probe_avg_mag_sqrd_c_sptr.**pc_work_time_avg**(*probe_avg_mag_sqrd_c_sptr self*) → float

probe_avg_mag_sqrd_c_sptr.**pc_work_time_total**(*probe_avg_mag_sqrd_c_sptr self*) → float

probe_avg_mag_sqrd_c_sptr.**reset**(*probe_avg_mag_sqrd_c_sptr self*)

probe_avg_mag_sqrd_c_sptr.**sample_delay**(*probe_avg_mag_sqrd_c_sptr self*, *int which*) → unsigned int

probe_avg_mag_sqrd_c_sptr.**set_alpha**(*probe_avg_mag_sqrd_c_sptr self*, *double alpha*)

probe_avg_mag_sqrd_c_sptr.**set_min_noutput_items**(*probe_avg_mag_sqrd_c_sptr self*, *int m*)

probe_avg_mag_sqrd_c_sptr.**set_thread_priority**(*probe_avg_mag_sqrd_c_sptr self*, *int priority*) → int

probe_avg_mag_sqrd_c_sptr.**set_threshold**(*probe_avg_mag_sqrd_c_sptr self*, *double decibels*)

probe_avg_mag_sqrd_c_sptr.**thread_priority**(*probe_avg_mag_sqrd_c_sptr self*) → int

probe_avg_mag_sqrd_c_sptr.**threshold**(*probe_avg_mag_sqrd_c_sptr self*) → double

probe_avg_mag_sqrd_c_sptr.**unmuted**(*probe_avg_mag_sqrd_c_sptr self*) → bool

gnuradio.analog.**probe_avg_mag_sqrd_cf**(*double threshold_db*, *double alpha=0.0001*) → probe_avg_mag_sqrd_cf_sptr

compute avg magnitude squared.

Input stream 0: complex Output stream 0: float

Compute a running average of the magnitude squared of the the input. The level and indication as to whether the level exceeds threshold can be retrieved with the level and unmuted accessors.

Constructor Specific Documentation:

Make a block that computes avg magnitude squared.

Parameters:
- **threshold_db** – Threshold for muting.
- **alpha** – Gain parameter for the running average filter.

probe_avg_mag_sqrd_cf_sptr.**active_thread_priority**(*probe_avg_mag_sqrd_cf_sptr*

self) → int

probe_avg_mag_sqrd_cf_sptr.**declare_sample_delay**(*probe_avg_mag_sqrd_cf_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_avg_mag_sqrd_cf_sptr self, unsigned int delay)

probe_avg_mag_sqrd_cf_sptr.**level**(*probe_avg_mag_sqrd_cf_sptr self*) → double

probe_avg_mag_sqrd_cf_sptr.**message_subscribers**(*probe_avg_mag_sqrd_cf_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_avg_mag_sqrd_cf_sptr.**min_noutput_items**(*probe_avg_mag_sqrd_cf_sptr self*) → int

probe_avg_mag_sqrd_cf_sptr.**pc_input_buffers_full_avg**(*probe_avg_mag_sqrd_cf_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_avg_mag_sqrd_cf_sptr self) -> pmt_vector_float

probe_avg_mag_sqrd_cf_sptr.**pc_noutput_items_avg**(*probe_avg_mag_sqrd_cf_sptr self*) → float

probe_avg_mag_sqrd_cf_sptr.**pc_nproduced_avg**(*probe_avg_mag_sqrd_cf_sptr self*) → float

probe_avg_mag_sqrd_cf_sptr.**pc_output_buffers_full_avg**(*probe_avg_mag_sqrd_cf_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_avg_mag_sqrd_cf_sptr self) -> pmt_vector_float

probe_avg_mag_sqrd_cf_sptr.**pc_throughput_avg**(*probe_avg_mag_sqrd_cf_sptr self*) → float

probe_avg_mag_sqrd_cf_sptr.**pc_work_time_avg**(*probe_avg_mag_sqrd_cf_sptr self*) → float

probe_avg_mag_sqrd_cf_sptr.**pc_work_time_total**(*probe_avg_mag_sqrd_cf_sptr self*) → float

probe_avg_mag_sqrd_cf_sptr.**reset**(*probe_avg_mag_sqrd_cf_sptr self*)

probe_avg_mag_sqrd_cf_sptr.**sample_delay**(*probe_avg_mag_sqrd_cf_sptr self*, *int which*) → unsigned int

probe_avg_mag_sqrd_cf_sptr.**set_alpha**(*probe_avg_mag_sqrd_cf_sptr self*, *double alpha*)

probe_avg_mag_sqrd_cf_sptr.**set_min_noutput_items**(*probe_avg_mag_sqrd_cf_sptr self*, *int m*)

probe_avg_mag_sqrd_cf_sptr.**set_thread_priority**(*probe_avg_mag_sqrd_cf_sptr self*, *int priority*) → int

probe_avg_mag_sqrd_cf_sptr.**set_threshold**(*probe_avg_mag_sqrd_cf_sptr self*, *double decibels*)

probe_avg_mag_sqrd_cf_sptr.**thread_priority**(*probe_avg_mag_sqrd_cf_sptr self*) → int

probe_avg_mag_sqrd_cf_sptr.**threshold**(*probe_avg_mag_sqrd_cf_sptr self*) → double

probe_avg_mag_sqrd_cf_sptr.**unmuted**(*probe_avg_mag_sqrd_cf_sptr self*) → bool

gnuradio.analog.**probe_avg_mag_sqrd_f**(*double threshold_db*, *double alpha=0.0001*) → probe_avg_mag_sqrd_f_sptr

compute avg magnitude squared.

input stream 0: float

Compute a running average of the magnitude squared of the the input. The level and indication as to whether the level exceeds threshold can be retrieved with the level and unmuted accessors.

Constructor Specific Documentation:

Make a float sink that computes avg magnitude squared.

Parameters:
- **threshold_db** – Threshold for muting.
- **alpha** – Gain parameter for the running average filter.

probe_avg_mag_sqrd_f_sptr.**active_thread_priority**(*probe_avg_mag_sqrd_f_sptr self*) → int

probe_avg_mag_sqrd_f_sptr.**declare_sample_delay**(*probe_avg_mag_sqrd_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_avg_mag_sqrd_f_sptr self, unsigned int delay)

probe_avg_mag_sqrd_f_sptr.**level**(*probe_avg_mag_sqrd_f_sptr self*) → double

probe_avg_mag_sqrd_f_sptr.**message_subscribers**(*probe_avg_mag_sqrd_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_avg_mag_sqrd_f_sptr.**min_noutput_items**(*probe_avg_mag_sqrd_f_sptr self*) → int

probe_avg_mag_sqrd_f_sptr.**pc_input_buffers_full_avg**(*probe_avg_mag_sqrd_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_avg_mag_sqrd_f_sptr self) -> pmt_vector_float

probe_avg_mag_sqrd_f_sptr.**pc_noutput_items_avg**(*probe_avg_mag_sqrd_f_sptr self*) → float

probe_avg_mag_sqrd_f_sptr.**pc_nproduced_avg**(*probe_avg_mag_sqrd_f_sptr self*) → float

probe_avg_mag_sqrd_f_sptr.**pc_output_buffers_full_avg**(*probe_avg_mag_sqrd_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_avg_mag_sqrd_f_sptr self) -> pmt_vector_float

probe_avg_mag_sqrd_f_sptr.**pc_throughput_avg**(*probe_avg_mag_sqrd_f_sptr self*) → float

probe_avg_mag_sqrd_f_sptr.**pc_work_time_avg**(*probe_avg_mag_sqrd_f_sptr self*) → float

probe_avg_mag_sqrd_f_sptr.**pc_work_time_total**(*probe_avg_mag_sqrd_f_sptr self*) → float

probe_avg_mag_sqrd_f_sptr.**reset**(*probe_avg_mag_sqrd_f_sptr self*)

probe_avg_mag_sqrd_f_sptr.**sample_delay**(*probe_avg_mag_sqrd_f_sptr self*, *int which*) → unsigned int

probe_avg_mag_sqrd_f_sptr.**set_alpha**(*probe_avg_mag_sqrd_f_sptr self*, *double alpha*)

probe_avg_mag_sqrd_f_sptr.**set_min_noutput_items**(*probe_avg_mag_sqrd_f_sptr self*, *int m*)

probe_avg_mag_sqrd_f_sptr.**set_thread_priority**(*probe_avg_mag_sqrd_f_sptr self*, *int priority*) → int

probe_avg_mag_sqrd_f_sptr.**set_threshold**(*probe_avg_mag_sqrd_f_sptr self*, *double decibels*)

probe_avg_mag_sqrd_f_sptr.**thread_priority**(*probe_avg_mag_sqrd_f_sptr self*) → int

probe_avg_mag_sqrd_f_sptr.**threshold**(*probe_avg_mag_sqrd_f_sptr self*) → double

probe_avg_mag_sqrd_f_sptr.**unmuted**(*probe_avg_mag_sqrd_f_sptr self*) → bool

gnuradio.analog.**pwr_squelch_cc**(*double db*, *double alpha=0.0001*, *int ramp=0*, *bool gate=False*) → pwr_squelch_cc_sptr

gate or zero output when input power below threshold

Constructor Specific Documentation:

Make power-based squelch block.

The block will emit a tag with the key pmt::intern("squelch_sob") with the value of pmt::PMT_NIL on the first item it passes, and with the key pmt::intern("squelch:eob") on the last item it passes.

Parameters:
- **db** – threshold (in dB) for power squelch
- **alpha** – Gain of averaging filter. Defaults to 0.0001.
- **ramp** – sets response characteristic. Defaults to 0.
- **gate** – if true, no output if no squelch tone. if false, output 0's if no squelch tone (default).

pwr_squelch_cc_sptr.**active_thread_priority**(*pwr_squelch_cc_sptr self*) → int

pwr_squelch_cc_sptr.**declare_sample_delay**(*pwr_squelch_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(pwr_squelch_cc_sptr self, unsigned int delay)

pwr_squelch_cc_sptr.**gate**(*pwr_squelch_cc_sptr self*) → bool

pwr_squelch_cc_sptr.**message_subscribers**(*pwr_squelch_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pwr_squelch_cc_sptr.**min_noutput_items**(*pwr_squelch_cc_sptr self*) → int

pwr_squelch_cc_sptr.**pc_input_buffers_full_avg**(*pwr_squelch_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(pwr_squelch_cc_sptr self) -> pmt_vector_float

pwr_squelch_cc_sptr.**pc_noutput_items_avg**(*pwr_squelch_cc_sptr self*) → float

pwr_squelch_cc_sptr.**pc_nproduced_avg**(*pwr_squelch_cc_sptr self*) → float

pwr_squelch_cc_sptr.**pc_output_buffers_full_avg**(*pwr_squelch_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(pwr_squelch_cc_sptr self) -> pmt_vector_float

pwr_squelch_cc_sptr.**pc_throughput_avg**(*pwr_squelch_cc_sptr self*) → float

pwr_squelch_cc_sptr.**pc_work_time_avg**(*pwr_squelch_cc_sptr self*) → float

pwr_squelch_cc_sptr.**pc_work_time_total**(*pwr_squelch_cc_sptr self*) → float

pwr_squelch_cc_sptr.**ramp**(*pwr_squelch_cc_sptr self*) → int

pwr_squelch_cc_sptr.**sample_delay**(*pwr_squelch_cc_sptr self*, *int which*) → unsigned int

pwr_squelch_cc_sptr.**set_alpha**(*pwr_squelch_cc_sptr self*, *double alpha*)

pwr_squelch_cc_sptr.**set_gate**(*pwr_squelch_cc_sptr self*, *bool gate*)

pwr_squelch_cc_sptr.**set_min_noutput_items**(*pwr_squelch_cc_sptr self*, *int m*)

pwr_squelch_cc_sptr.**set_ramp**(*pwr_squelch_cc_sptr self*, *int ramp*)

pwr_squelch_cc_sptr.**set_thread_priority**(*pwr_squelch_cc_sptr self*, *int priority*) → int

pwr_squelch_cc_sptr.**set_threshold**(*pwr_squelch_cc_sptr self*, *double db*)

pwr_squelch_cc_sptr.**squelch_range**(*pwr_squelch_cc_sptr self*) → pmt_vector_float

pwr_squelch_cc_sptr.**thread_priority**(*pwr_squelch_cc_sptr self*) → int

pwr_squelch_cc_sptr.**threshold**(*pwr_squelch_cc_sptr self*) → double

pwr_squelch_cc_sptr.**unmuted**(*pwr_squelch_cc_sptr self*) → bool

gnuradio.analog.**pwr_squelch_ff**(*double db*, *double alpha=0.0001*, *int ramp=0*, *bool gate=False*) → pwr_squelch_ff_sptr

gate or zero output when input power below threshold

Constructor Specific Documentation:

Make power-based squelch block.

The block will emit a tag with the key pmt::intern("squelch_sob") with the value of pmt::PMT_NIL on the first item it passes, and with the key pmt::intern("squelch:eob") on the last item it passes.

Parameters:
- **db** – threshold (in dB) for power squelch
- **alpha** – Gain of averaging filter. Defaults to 0.0001.
- **ramp** – sets response characteristic. Defaults to 0.
- **gate** – if true, no output if no squelch tone. if false, output 0's if no squelch tone (default).

pwr_squelch_ff_sptr.**active_thread_priority**(*pwr_squelch_ff_sptr self*) → int

pwr_squelch_ff_sptr.**declare_sample_delay**(*pwr_squelch_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(pwr_squelch_ff_sptr self, unsigned int delay)

pwr_squelch_ff_sptr.**gate**(*pwr_squelch_ff_sptr self*) → bool

pwr_squelch_ff_sptr.**message_subscribers**(*pwr_squelch_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pwr_squelch_ff_sptr.**min_noutput_items**(*pwr_squelch_ff_sptr self*) → int

pwr_squelch_ff_sptr.**pc_input_buffers_full_avg**(*pwr_squelch_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(pwr_squelch_ff_sptr self) -> pmt_vector_float

pwr_squelch_ff_sptr.**pc_noutput_items_avg**(*pwr_squelch_ff_sptr self*) → float

pwr_squelch_ff_sptr.**pc_nproduced_avg**(*pwr_squelch_ff_sptr self*) → float

pwr_squelch_ff_sptr.**pc_output_buffers_full_avg**(*pwr_squelch_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(pwr_squelch_ff_sptr self) -> pmt_vector_float

pwr_squelch_ff_sptr.**pc_throughput_avg**(*pwr_squelch_ff_sptr self*) → float

pwr_squelch_ff_sptr.**pc_work_time_avg**(*pwr_squelch_ff_sptr self*) → float

pwr_squelch_ff_sptr.**pc_work_time_total**(*pwr_squelch_ff_sptr self*) → float

pwr_squelch_ff_sptr.**ramp**(*pwr_squelch_ff_sptr self*) → int

pwr_squelch_ff_sptr.**sample_delay**(*pwr_squelch_ff_sptr self*, *int which*) → unsigned int

pwr_squelch_ff_sptr.**set_alpha**(*pwr_squelch_ff_sptr self*, *double alpha*)

pwr_squelch_ff_sptr.**set_gate**(*pwr_squelch_ff_sptr self*, *bool gate*)

pwr_squelch_ff_sptr.**set_min_noutput_items**(*pwr_squelch_ff_sptr self*, *int m*)

pwr_squelch_ff_sptr.**set_ramp**(*pwr_squelch_ff_sptr self*, *int ramp*)

pwr_squelch_ff_sptr.**set_thread_priority**(*pwr_squelch_ff_sptr self*, *int priority*) → int

pwr_squelch_ff_sptr.**set_threshold**(*pwr_squelch_ff_sptr self*, *double db*)

pwr_squelch_ff_sptr.**squelch_range**(*pwr_squelch_ff_sptr self*) → pmt_vector_float

pwr_squelch_ff_sptr.**thread_priority**(*pwr_squelch_ff_sptr self*) → int

pwr_squelch_ff_sptr.**threshold**(*pwr_squelch_ff_sptr self*) → double

pwr_squelch_ff_sptr.**unmuted**(*pwr_squelch_ff_sptr self*) → bool

gnuradio.analog.**quadrature_demod_cf**(*float gain*) → quadrature_demod_cf_sptr

quadrature demodulator: complex in, float out

This can be used to demod FM, FSK, GMSK, etc. The input is complex baseband, output is the signal frequency in relation to the sample rated, multiplied with the gain.

Mathematically, this block calculates the product of the one-sample delayed input and the conjugate undelayed signal, and then calculates the argument of the resulting complex number:

.

Let be a complex sinusoid with amplitude , (absolute) frequency and phase sampled at so, without loss of generality,

then

Constructor Specific Documentation:

Parameters: **gain** –

quadrature_demod_cf_sptr.**active_thread_priority**(*quadrature_demod_cf_sptr self*) → int

quadrature_demod_cf_sptr.**declare_sample_delay**(*quadrature_demod_cf_sptr self*, *int which*, *int delay*)

declare_sample_delay(quadrature_demod_cf_sptr self, unsigned int delay)

quadrature_demod_cf_sptr.**gain**(*quadrature_demod_cf_sptr self*) → float

quadrature_demod_cf_sptr.**message_subscribers**(*quadrature_demod_cf_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

quadrature_demod_cf_sptr.**min_noutput_items**(*quadrature_demod_cf_sptr self*) → int

quadrature_demod_cf_sptr.**pc_input_buffers_full_avg**(*quadrature_demod_cf_sptr self*, *int which*) → float

pc_input_buffers_full_avg(quadrature_demod_cf_sptr self) -> pmt_vector_float

quadrature_demod_cf_sptr.**pc_noutput_items_avg**(*quadrature_demod_cf_sptr self*) → float

quadrature_demod_cf_sptr.**pc_nproduced_avg**(*quadrature_demod_cf_sptr self*) → float

quadrature_demod_cf_sptr.**pc_output_buffers_full_avg**(*quadrature_demod_cf_sptr self*, *int which*) → float

pc_output_buffers_full_avg(quadrature_demod_cf_sptr self) -> pmt_vector_float

quadrature_demod_cf_sptr.**pc_throughput_avg**(*quadrature_demod_cf_sptr self*) → float

quadrature_demod_cf_sptr.**pc_work_time_avg**(*quadrature_demod_cf_sptr self*) → float

quadrature_demod_cf_sptr.**pc_work_time_total**(*quadrature_demod_cf_sptr self*) → float

quadrature_demod_cf_sptr.**sample_delay**(*quadrature_demod_cf_sptr self*, *int which*) → unsigned int

quadrature_demod_cf_sptr.**set_gain**(*quadrature_demod_cf_sptr self*, *float gain*)

quadrature_demod_cf_sptr.**set_min_noutput_items**(*quadrature_demod_cf_sptr self*, *int m*)

quadrature_demod_cf_sptr.**set_thread_priority**(*quadrature_demod_cf_sptr self*, *int priority*) → int

quadrature_demod_cf_sptr.**thread_priority**(*quadrature_demod_cf_sptr self*) → int

gnuradio.analog.**rail_ff**(*float lo*, *float hi*) → rail_ff_sptr

clips input values to min, max

Constructor Specific Documentation:

Build a rail block.

Parameters:
- **lo** – the low value to clip to.
- **hi** – the high value to clip to.

rail_ff_sptr.**active_thread_priority**(*rail_ff_sptr self*) → int

rail_ff_sptr.**declare_sample_delay**(*rail_ff_sptr self*, *int which*, *int delay*)
declare_sample_delay(rail_ff_sptr self, unsigned int delay)

rail_ff_sptr.**hi**(*rail_ff_sptr self*) → float

rail_ff_sptr.**lo**(*rail_ff_sptr self*) → float

rail_ff_sptr.**message_subscribers**(*rail_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

rail_ff_sptr.**min_noutput_items**(*rail_ff_sptr self*) → int

rail_ff_sptr.**pc_input_buffers_full_avg**(*rail_ff_sptr self*, *int which*) → float
pc_input_buffers_full_avg(rail_ff_sptr self) -> pmt_vector_float

rail_ff_sptr.**pc_noutput_items_avg**(*rail_ff_sptr self*) → float

rail_ff_sptr.**pc_nproduced_avg**(*rail_ff_sptr self*) → float

rail_ff_sptr.**pc_output_buffers_full_avg**(*rail_ff_sptr self*, *int which*) → float
pc_output_buffers_full_avg(rail_ff_sptr self) -> pmt_vector_float

rail_ff_sptr.**pc_throughput_avg**(*rail_ff_sptr self*) → float

rail_ff_sptr.**pc_work_time_avg**(*rail_ff_sptr self*) → float

rail_ff_sptr.**pc_work_time_total**(*rail_ff_sptr self*) → float

rail_ff_sptr.**sample_delay**(*rail_ff_sptr self*, *int which*) → unsigned int

rail_ff_sptr.**set_hi**(*rail_ff_sptr self*, *float hi*)

rail_ff_sptr.**set_lo**(*rail_ff_sptr self*, *float lo*)

rail_ff_sptr.**set_min_noutput_items**(*rail_ff_sptr self*, *int m*)

rail_ff_sptr.**set_thread_priority**(*rail_ff_sptr self*, *int priority*) → int

rail_ff_sptr.**thread_priority**(*rail_ff_sptr self*) → int

gnuradio.analog.**sig_source_c**(*double sampling_freq*, *gr::analog::gr_waveform_t waveform*, *double wave_freq*, *double ampl*, *gr_complex offset=0*) → sig_source_c_sptr
signal generator with gr_complex output.

Constructor Specific Documentation:

Build a signal source block.

Parameters:
- **sampling_freq** – Sampling rate of signal.
- **waveform** – wavetform type.
- **wave_freq** – Frequency of waveform (relative to sampling_freq).
- **ampl** – Signal amplitude.
- **offset** – offset of signal.

sig_source_c_sptr.**active_thread_priority**(*sig_source_c_sptr self*) → int

sig_source_c_sptr.**amplitude**(*sig_source_c_sptr self*) → double

sig_source_c_sptr.**declare_sample_delay**(*sig_source_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(sig_source_c_sptr self, unsigned int delay)

sig_source_c_sptr.**frequency**(*sig_source_c_sptr self*) → double

sig_source_c_sptr.**message_subscribers**(*sig_source_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sig_source_c_sptr.**min_noutput_items**(*sig_source_c_sptr self*) → int

sig_source_c_sptr.**offset**(*sig_source_c_sptr self*) → gr_complex

sig_source_c_sptr.**pc_input_buffers_full_avg**(*sig_source_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sig_source_c_sptr self) -> pmt_vector_float

sig_source_c_sptr.**pc_noutput_items_avg**(*sig_source_c_sptr self*) → float

sig_source_c_sptr.**pc_nproduced_avg**(*sig_source_c_sptr self*) → float

sig_source_c_sptr.**pc_output_buffers_full_avg**(*sig_source_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sig_source_c_sptr self) -> pmt_vector_float

sig_source_c_sptr.**pc_throughput_avg**(*sig_source_c_sptr self*) → float

sig_source_c_sptr.**pc_work_time_avg**(*sig_source_c_sptr self*) → float

sig_source_c_sptr.**pc_work_time_total**(*sig_source_c_sptr self*) → float

sig_source_c_sptr.**sample_delay**(*sig_source_c_sptr self*, *int which*) → unsigned int

sig_source_c_sptr.**sampling_freq**(*sig_source_c_sptr self*) → double

sig_source_c_sptr.**set_amplitude**(*sig_source_c_sptr self*, *double ampl*)

sig_source_c_sptr.**set_frequency**(*sig_source_c_sptr self*, *double frequency*)

sig_source_c_sptr.**set_min_noutput_items**(*sig_source_c_sptr self*, *int m*)

sig_source_c_sptr.**set_offset**(*sig_source_c_sptr self*, *gr_complex offset*)

sig_source_c_sptr.**set_sampling_freq**(*sig_source_c_sptr self*, *double sampling_freq*)

sig_source_c_sptr.**set_thread_priority**(*sig_source_c_sptr self*, *int priority*) → int

sig_source_c_sptr.**set_waveform**(*sig_source_c_sptr self*, *gr::analog::gr_waveform_t waveform*)

sig_source_c_sptr.**thread_priority**(*sig_source_c_sptr self*) → int

sig_source_c_sptr.**waveform**(*sig_source_c_sptr self*) → gr::analog::gr_waveform_t

gnuradio.analog.**sig_source_f**(*double sampling_freq*, *gr::analog::gr_waveform_t waveform*, *double wave_freq*, *double ampl*, *float offset=0*) → sig_source_f_sptr

signal generator with float output.

Constructor Specific Documentation:

Build a signal source block.

Parameters:
- **sampling_freq** – Sampling rate of signal.
- **waveform** – wavetform type.
- **wave_freq** – Frequency of waveform (relative to sampling_freq).
- **ampl** – Signal amplitude.
- **offset** – offset of signal.

sig_source_f_sptr.**active_thread_priority**(*sig_source_f_sptr self*) → int

sig_source_f_sptr.**amplitude**(*sig_source_f_sptr self*) → double

sig_source_f_sptr.**declare_sample_delay**(*sig_source_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(sig_source_f_sptr self, unsigned int delay)

sig_source_f_sptr.**frequency**(*sig_source_f_sptr self*) → double

sig_source_f_sptr.**message_subscribers**(*sig_source_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sig_source_f_sptr.**min_noutput_items**(*sig_source_f_sptr self*) → int

sig_source_f_sptr.**offset**(*sig_source_f_sptr self*) → float

sig_source_f_sptr.**pc_input_buffers_full_avg**(*sig_source_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sig_source_f_sptr self) -> pmt_vector_float

sig_source_f_sptr.**pc_noutput_items_avg**(*sig_source_f_sptr self*) → float

sig_source_f_sptr.**pc_nproduced_avg**(*sig_source_f_sptr self*) → float

sig_source_f_sptr.**pc_output_buffers_full_avg**(*sig_source_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sig_source_f_sptr self) -> pmt_vector_float

sig_source_f_sptr.**pc_throughput_avg**(*sig_source_f_sptr self*) → float

sig_source_f_sptr.**pc_work_time_avg**(*sig_source_f_sptr self*) → float

sig_source_f_sptr.**pc_work_time_total**(*sig_source_f_sptr self*) → float

sig_source_f_sptr.**sample_delay**(*sig_source_f_sptr self*, *int which*) → unsigned int

sig_source_f_sptr.**sampling_freq**(*sig_source_f_sptr self*) → double

sig_source_f_sptr.**set_amplitude**(*sig_source_f_sptr self*, *double ampl*)

sig_source_f_sptr.**set_frequency**(*sig_source_f_sptr self*, *double frequency*)

sig_source_f_sptr.**set_min_noutput_items**(*sig_source_f_sptr self*, *int m*)

sig_source_f_sptr.**set_offset**(*sig_source_f_sptr self*, *float offset*)

sig_source_f_sptr.**set_sampling_freq**(*sig_source_f_sptr self*, *double sampling_freq*)

sig_source_f_sptr.**set_thread_priority**(*sig_source_f_sptr self*, *int priority*) → int

sig_source_f_sptr.**set_waveform**(*sig_source_f_sptr self*, *gr::analog::gr_waveform_t waveform*)

sig_source_f_sptr.**thread_priority**(*sig_source_f_sptr self*) → int

sig_source_f_sptr.**waveform**(*sig_source_f_sptr self*) → gr::analog::gr_waveform_t

gnuradio.analog.**sig_source_i**(*double sampling_freq*, *gr::analog::gr_waveform_t waveform*, *double wave_freq*, *double ampl*, *int offset=0*) → sig_source_i_sptr

signal generator with int output.

Constructor Specific Documentation:

Build a signal source block.

Parameters:
- **sampling_freq** – Sampling rate of signal.
- **waveform** – wavetform type.
- **wave_freq** – Frequency of waveform (relative to sampling_freq).
- **ampl** – Signal amplitude.
- **offset** – offset of signal.

sig_source_i_sptr.**active_thread_priority**(*sig_source_i_sptr self*) → int

sig_source_i_sptr.**amplitude**(*sig_source_i_sptr self*) → double

sig_source_i_sptr.**declare_sample_delay**(*sig_source_i_sptr self*, *int which*, *int delay*)

declare_sample_delay(sig_source_i_sptr self, unsigned int delay)

sig_source_i_sptr.**frequency**(*sig_source_i_sptr self*) → double

sig_source_i_sptr.**message_subscribers**(*sig_source_i_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sig_source_i_sptr.**min_noutput_items**(*sig_source_i_sptr self*) → int

sig_source_i_sptr.**offset**(*sig_source_i_sptr self*) → int

sig_source_i_sptr.**pc_input_buffers_full_avg**(*sig_source_i_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sig_source_i_sptr self) -> pmt_vector_float

sig_source_i_sptr.**pc_noutput_items_avg**(*sig_source_i_sptr self*) → float

sig_source_i_sptr.**pc_nproduced_avg**(*sig_source_i_sptr self*) → float

sig_source_i_sptr.**pc_output_buffers_full_avg**(*sig_source_i_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sig_source_i_sptr self) -> pmt_vector_float

sig_source_i_sptr.**pc_throughput_avg**(*sig_source_i_sptr self*) → float

sig_source_i_sptr.**pc_work_time_avg**(*sig_source_i_sptr self*) → float

sig_source_i_sptr.**pc_work_time_total**(*sig_source_i_sptr self*) → float

sig_source_i_sptr.**sample_delay**(*sig_source_i_sptr self*, *int which*) → unsigned int

sig_source_i_sptr.**sampling_freq**(*sig_source_i_sptr self*) → double

sig_source_i_sptr.**set_amplitude**(*sig_source_i_sptr self*, *double ampl*)

sig_source_i_sptr.**set_frequency**(*sig_source_i_sptr self*, *double frequency*)

sig_source_i_sptr.**set_min_noutput_items**(*sig_source_i_sptr self*, *int m*)

sig_source_i_sptr.**set_offset**(*sig_source_i_sptr self*, *int offset*)

sig_source_i_sptr.**set_sampling_freq**(*sig_source_i_sptr self*, *double sampling_freq*)

sig_source_i_sptr.**set_thread_priority**(*sig_source_i_sptr self*, *int priority*) → int

sig_source_i_sptr.**set_waveform**(*sig_source_i_sptr self*, *gr::analog::gr_waveform_t waveform*)

sig_source_i_sptr.**thread_priority**(*sig_source_i_sptr self*) → int

sig_source_i_sptr.**waveform**(*sig_source_i_sptr self*) → gr::analog::gr_waveform_t

gnuradio.analog.**sig_source_s**(*double sampling_freq*, *gr::analog::gr_waveform_t waveform*, *double wave_freq*, *double ampl*, *short offset=0*) → sig_source_s_sptr

signal generator with short output.

Constructor Specific Documentation:

Build a signal source block.

Parameters:

- **sampling_freq** – Sampling rate of signal.
- **waveform** – wavetform type.
- **wave_freq** – Frequency of waveform (relative to sampling_freq).
- **ampl** – Signal amplitude.
- **offset** – offset of signal.

sig_source_s_sptr.**active_thread_priority**(*sig_source_s_sptr self*) → int

sig_source_s_sptr.**amplitude**(*sig_source_s_sptr self*) → double

sig_source_s_sptr.**declare_sample_delay**(*sig_source_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(sig_source_s_sptr self, unsigned int delay)

sig_source_s_sptr.**frequency**(*sig_source_s_sptr self*) → double

sig_source_s_sptr.**message_subscribers**(*sig_source_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sig_source_s_sptr.**min_noutput_items**(*sig_source_s_sptr self*) → int

sig_source_s_sptr.**offset**(*sig_source_s_sptr self*) → short

sig_source_s_sptr.**pc_input_buffers_full_avg**(*sig_source_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sig_source_s_sptr self) -> pmt_vector_float

sig_source_s_sptr.**pc_noutput_items_avg**(*sig_source_s_sptr self*) → float

sig_source_s_sptr.**pc_nproduced_avg**(*sig_source_s_sptr self*) → float

sig_source_s_sptr.**pc_output_buffers_full_avg**(*sig_source_s_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sig_source_s_sptr self) -> pmt_vector_float

sig_source_s_sptr.**pc_throughput_avg**(*sig_source_s_sptr self*) → float

sig_source_s_sptr.**pc_work_time_avg**(*sig_source_s_sptr self*) → float

sig_source_s_sptr.**pc_work_time_total**(*sig_source_s_sptr self*) → float

sig_source_s_sptr.**sample_delay**(*sig_source_s_sptr self*, *int which*) → unsigned int

sig_source_s_sptr.**sampling_freq**(*sig_source_s_sptr self*) → double

sig_source_s_sptr.**set_amplitude**(*sig_source_s_sptr self*, *double ampl*)

sig_source_s_sptr.**set_frequency**(*sig_source_s_sptr self*, *double frequency*)

sig_source_s_sptr.**set_min_noutput_items**(*sig_source_s_sptr self*, *int m*)

sig_source_s_sptr.**set_offset**(*sig_source_s_sptr self*, *short offset*)

sig_source_s_sptr.**set_sampling_freq**(*sig_source_s_sptr self*, *double sampling_freq*)

sig_source_s_sptr.**set_thread_priority**(*sig_source_s_sptr self*, *int priority*) → int

sig_source_s_sptr.**set_waveform**(*sig_source_s_sptr self*, *gr::analog::gr_waveform_t waveform*)

sig_source_s_sptr.**thread_priority**(*sig_source_s_sptr self*) → int

sig_source_s_sptr.**waveform**(*sig_source_s_sptr self*) → gr::analog::gr_waveform_t

gnuradio.analog.**simple_squelch_cc**(*double threshold_db*, *double alpha*) → simple_squelch_cc_sptr

simple squelch block based on average signal power and threshold in dB.

Constructor Specific Documentation:

Make a simple squelch block.

Parameters:
- **threshold_db** – Threshold for muting.
- **alpha** – Gain parameter for the running average filter.

simple_squelch_cc_sptr.**active_thread_priority**(*simple_squelch_cc_sptr self*) → int

simple_squelch_cc_sptr.**declare_sample_delay**(*simple_squelch_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(simple_squelch_cc_sptr self, unsigned int delay)

simple_squelch_cc_sptr.**message_subscribers**(*simple_squelch_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

simple_squelch_cc_sptr.**min_noutput_items**(*simple_squelch_cc_sptr self*) → int

simple_squelch_cc_sptr.**pc_input_buffers_full_avg**(*simple_squelch_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(simple_squelch_cc_sptr self) -> pmt_vector_float

simple_squelch_cc_sptr.**pc_noutput_items_avg**(*simple_squelch_cc_sptr self*) → float

simple_squelch_cc_sptr.**pc_nproduced_avg**(*simple_squelch_cc_sptr self*) → float

simple_squelch_cc_sptr.**pc_output_buffers_full_avg**(*simple_squelch_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(simple_squelch_cc_sptr self) -> pmt_vector_float

simple_squelch_cc_sptr.**pc_throughput_avg**(*simple_squelch_cc_sptr self*) → float

simple_squelch_cc_sptr.**pc_work_time_avg**(*simple_squelch_cc_sptr self*) → float

simple_squelch_cc_sptr.**pc_work_time_total**(*simple_squelch_cc_sptr self*) → float

simple_squelch_cc_sptr.**sample_delay**(*simple_squelch_cc_sptr self*, *int which*) → unsigned int

simple_squelch_cc_sptr.**set_alpha**(*simple_squelch_cc_sptr self*, *double alpha*)

simple_squelch_cc_sptr.**set_min_noutput_items**(*simple_squelch_cc_sptr self*, *int m*)

simple_squelch_cc_sptr.**set_thread_priority**(*simple_squelch_cc_sptr self*, *int priority*) → int

simple_squelch_cc_sptr.**set_threshold**(*simple_squelch_cc_sptr self*, *double decibels*)

simple_squelch_cc_sptr.**squelch_range**(*simple_squelch_cc_sptr self*) → pmt_vector_float

simple_squelch_cc_sptr.**thread_priority**(*simple_squelch_cc_sptr self*) → int

simple_squelch_cc_sptr.**threshold**(*simple_squelch_cc_sptr self*) → double

simple_squelch_cc_sptr.**unmuted**(*simple_squelch_cc_sptr self*) → bool