Simple User Manual for Gnuradio 3.1.1

This Document Was Last Updated on:

25-November-2007

By:

Firas Abbas

Table of Contents

1) gnuradio package

Description	The main package. All gnuradio stuff are here

1.1) gnuradio/blks sub package

Type	Folder
Description	Semi-hideous kludge to import everything in the blksimpl directory into the gnuradio.blks namespace. This keeps us from having to remember to manually update this file.

1.1.1) gnuradio/blksimpl/am_demod.py

Type	Python file
Description	AM demodulation block
Examples	
Note	

1.1.1.1) am_demod_cf ( )

Type	Function
Description	Generalized AM demodulation block with audio filtering.This block demodulates a band-limited, complex down-converted AM channel into the original baseband signal, applying low pass filtering to the audio output. It produces a float stream in the range [-1.0, +1.0].
Usage	**blks.am_demod_cf (fg, channel_rate, audio_decim, audio_pass, audio_stop)**
Parameters	**fg**: flowgraph **channel_rate**: incoming sample rate of the AM baseband type channel_rate: integer **audio_decim**: input to output decimation rate type audio_decim: integer **audio_pass**: audio low pass filter passband frequency type audio_pass: float **audio_stop**: audio low pass filter stop frequency type audio_stop: float

1.1.1.2) demod_10k0a3e_cf ()

Type	Function
Description	AM demodulation block, 10 KHz channel. This block demodulates an AM channel conformant to 10K0A3E emission standards, such as broadcast band AM transmissions.
Usage	**blks.demod_10k0a3e_cf(fg, channel_rate, audio_decim)**
Parameters	**fg**: flowgraph **channel_rate**: incoming sample rate of the AM baseband type channel_rate: integer **audio_decim**: input to output decimation rate type audio_decim: integer

1.1.2) gnuradio/blksimpl/channel_model.py

Type	Python file
Description	Creates a channel model
Examples	
Note	

1.1.2.1) channel_model ( )

Type	Function
Description	Creates a channel model that includes: - AWGN noise power in terms of noise voltage - A frequency offset in the channel in ratio - A timing offset ratio to model clock difference (clock rate ratio) (epsilon). It is sample rate difference between tx and rx - Multipath taps
Usage	**blks.channel_model(fg, noise_voltage=0.0, frequency_offset=0.0, epsilon=1.0, taps=[1.0,0.0])**
Parameters	
Sub Function 1	blks.channel_model.set_noise_voltage(noise_voltage)
Sub Function 2	blks.channel_model.set_frequency_offset(frequency_offset)
Sub Function 3	blks.channel_model.set_taps(taps)

1.1.3) gnuradio/blksimpl/cpm.py

Type	Python file
Description	Continuous Phase modulation.
Examples	See gnuradio-examples/python/digital for examples
Note	

1.1.3.1) cpm_mod ( )

Type	Function
Description	Hierarchical block for Continuous Phase Modulation. The input is a byte stream (unsigned char) representing packed bits and the output is the complex modulated signal at baseband. See Proakis for definition of generic CPM signals: s(t)=exp(j phi(t)) phi(t)= 2 pi h int_0^t f(t') dt' f(t)=sum_k a_k g(t-kT) (normalizing assumption: int_0^infty g(t) dt = 1/2)
Usage	**blks.cpm_mod(fg,samples_per_symbol=2,bits_per_symbol=1, h_numerator=1, h_denominator=2, cpm_type=0,bt=_def_bt, symbols_per_pulse=1, generic_taps numpy.empty(1), verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per baud >= 2 type samples_per_symbol: integer **bits_per_symbol**: bits per symbol type bits_per_symbol: integer **h_numerator**: numerator of modulation index type h_numerator: integer **h_denominator**: denominator of modulation index (numerator and denominator must be relative primes) type h_denominator: integer **cpm_type**: supported types are: 0=CPFSK, 1=GMSK, 2=RC, 3=GENERAL type cpm_type: integer **bt**: bandwidth symbol time product for GMSK type bt: float **symbols_per_pulse**: shaping pulse duration in symbols type symbols_per_pulse: integer **generic_taps**: define a generic CPM pulse shape (sum = samples_per_symbol/2)

	type generic_taps: array of floats **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.1.4) gnuradio/blksimpl/d8psk.py

Type	Python file
Description	differential 8PSK modulation and demodulation
Examples	See gnuradio-examples/python/digital for examples
Note	

1.1.4.1) d8psk_mod ( )

Type	Function , D8PSK modulator
Description	Hierarchical block for RRC-filtered QPSK modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks.d8psk_mod(fg, samples_per_symbol=3, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.1.4.2) d8psk_demod ( )

Type	Function , D8PSK demodulator
Description	Differentially coherent detection of differentially encoded 8psk. Hierarchical block for RRC-filtered DQPSK demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (LSB)
Usage	**blks.d8psk_demod(fg, samples_per_symbol=3, excess_bw=.35, costas_alpha=.175, gain_mu=.175, mu=0.5, omega_relative_limit=.005, gray_code=True, verbose=False, log=False)**

Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: float **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **costas_alpha**: loop filter gain type costas_alpha: float **gain_mu**: for M&M block type gain_mu: float **mu**: for M&M block type mu: float **omega_relative_limit**: for M&M block type omega_relative_limit: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print demodulation data to files? type debug: bool

1.1.5) gnuradio/blksimpl/dbpsk.py

Type	Python file
Description	differential BPSK modulation and demodulation
Examples	See gnuradio-examples/python/digital for examples
Note	

1.1.5.1) dbpsk_mod ( )

Type	Function , DBPSK modulator
Description	Hierarchical block for RRC-filtered BPSK modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks.dbpsk_mod(fg, samples_per_symbol=2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per baud >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **log**: Log modulation data to files? type log: bool

1.1.5.2) dbpsk_demod ( )

Type	Function , DBPSK demodulator
Description	Differentially coherent detection of differentially encoded BPSK. Hierarchical block for RRC-filtered DBPSK demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (LSB).
Usage	**blks.d8psk_demod(fg, samples_per_symbol=2, excess_bw=.35, costas_alpha=.1, gain_mu=None, mu=0.5, omega_relative_limit=.005, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: float **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **costas_alpha**: loop filter gain type costas_alpha: float **gain_mu**: for M&M block type gain_mu: float **mu**: for M&M block type mu: float **omega_relative_limit**: for M&M block type omega_relative_limit: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print demodulation data to files? type debug: bool

1.1.6) gnuradio/blksimpl/dqpsk.py

Type	Python file
Description	differential QPSK modulation and demodulation
Examples	See gnuradio-examples/python/digital for examples
Note	

1.1.6.1) dqpsk_mod ( )

Type	Function , DQPSK modulator
Description	Hierarchical block for RRC-filtered QPSK modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks.dqpsk_mod(fg, samples_per_symbol=2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph

	type fg: flow graph **samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.1.6.2) dqpsk_demod ( )

Type	Function , DQPSK demodulator
Description	Differentially coherent detection of differentially encoded QPSK. Hierarchical block for RRC-filtered DQPSK demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (LSB)
Usage	**blks.d8psk_demod(fg, samples_per_symbol=2, excess_bw=.35, costas_alpha=.15, gain_mu=None, mu=0.5, omega_relative_limit=.005, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: float **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **costas_alpha**: loop filter gain type costas_alpha: float **gain_mu**: for M&M block type gain_mu: float **mu**: for M&M block type mu: float **omega_relative_limit**: for M&M block type omega_relative_limit: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.1.7) gnuradio/blksimpl/filterbank.py

Type	Python file
Description	Include Both Filter Synthesis and Analysis
Examples	See gnuradio-examples/python/usrp folder
Note	

1.1.7.1) synthesis_filterbank ( )

Type	Function

Description	Uniformly modulated polyphase DFT filter bank: synthesis
	See http://cnx.rice.edu/content/m10424/latest
	Takes M complex streams in, produces single complex stream out that runs at M times the input sample rate
	The channel spacing is equal to the input sample rate.
	The total bandwidth and output sample rate are equal the
	input sample rate * nchannels.
	Output stream to frequency mapping:
	channel zero is at zero frequency.
	
	if mpoints is odd:
	
	Channels with increasing positive frequencies come from
	channels 1 through (N-1)/2.
	Channel (N+1)/2 is the maximum negative frequency, and
	frequency increases through N-1 which is one channel lower
	than the zero frequency.
	
	if mpoints is even:
	
	Channels with increasing positive frequencies come from
	channels 1 through (N/2)-1.
	Channel (N/2) is evenly split between the max positive
	and negative bins.
	
	Channel (N/2)+1 is the maximum negative frequency, and
	frequency increases through N-1 which is one channel lower
	than the zero frequency.
	Channels near the frequency extremes end up getting cut
	off by subsequent filters and therefore have diminished
	utility.

Usage	**blks.synthesis_filterbank (fg, mpoints, taps=None)**
Parameters	**fg**: flow_graph **mpoints**: number of freq bins/interpolation factor/subbands **taps**: filter taps for subband filter
Example	See ayfabtu.py

1.1.7.2) analysis_filterbank ( )

Type	Function
Description	Uniformly modulated polyphase DFT filter bank: analysis See http://cnx.rice.edu/content/m10424/latest Takes 1 complex stream in, produces M complex streams out that runs at 1/M times the input sample rate. Same channel to frequency mapping as described in filter synthesis.
Usage	**blks.analysis_filterbank (fg, mpoints,taps)**
Parameters	**fg**: flow_graph **mpoints**: number of freq bins/interpolation factor/subbands **taps**: filter taps for subband filter
Examples	See test_dft_analysis

1.1.8) gnuradio/blksimpl/fm_demod.py

Type	Python file
Description	FM demodulation block
Examples	
Note	

1.1.8.1) fm_demod_cf ( )

Type	Function
Description	Generalized FM demodulation block with deemphasis and audio filtering.This block demodulates a band-limited, complex down-converted FM channel into the original baseband signal, optionally applying deemphasis. Low pass filtering is done on the resultant signal. It produces an output float stream in the range of [-1.0, +1.0].
Usage	**blks.fm_demod_cf (fg, channel_rate, audio_decim, deviation, audio_pass, audio_stop, gain=1.0, tau=75e-6)**
Parameters	**fg**: flowgraph **channel_rate**: incoming sample rate of the FM baseband type sample_rate: integer **deviation**: maximum FM deviation (default = 5000) type deviation: float **audio_decim**: input to output decimation rate type audio_decim: integer **audio_pass**: audio low pass filter passband frequency type audio_pass: float **audio_stop**: audio low pass filter stop frequency type audio_stop: float **gain**: gain applied to audio output (default = 1.0) type gain: float **tau**: deemphasis time constant (default = 75e-6), specify 'None' to prevent deemphasis

1.1.8.2) demod_20k0f3e_cf ()

Type	Function
Description	NBFM demodulation block, 20 KHz channels
Usage	**blks.demod_20k0f3e_cf(fg, channel_rate, audio_decim)**
Parameters	**fg**: flowgraph **sample_rate**: incoming sample rate of the FM baseband type sample_rate: integer **audio_decim**: input to output decimation rate type audio_decim: integer

1.1.8.3) demod_200kf3e_cf ()

Type	Function
Description	WFM demodulation block, mono. This block demodulates a complex, down converted, wideband FM channel conforming to 200KF3E emission standards, outputting floats in the range [-1.0, +1.0].
Usage	**blks.demod_200kf3e_cf(fg, channel_rate, audio_decim)**
Parameters	**fg**: flowgraph **sample_rate**: incoming sample rate of the FM baseband type sample_rate: integer **audio_decim**: input to output decimation rate type audio_decim: integer

1.1.9) gnuradio/blksimpl/fm_emph.py

Type	Python file
Description	FM deemphasis and preemphasis IIR filter
Examples	
Note	

1.1.9.1) fm_deemph ( )

<table>
<tr><td>Type</td><td>Function</td></tr>
<tr><td>Description</td><td>FM Deemphasis IIR filter.
         1
H(s) = -------
       1 + s
tau is the RC time constant.
critical frequency: w_p = 1/tau
We prewarp and use the bilinear z-transform to get our IIR coefficients.
See "Digital Signal Processing: A Practical Approach" by Ifeachor and Jervis</td></tr>
<tr><td>Usage</td><td>blks.fm_deemph(fg, fs, tau=75e-6)</td></tr>
<tr><td>Parameters</td><td>fg: flow graph
type fg: gr.flow_graph
fs: sampling frequency in Hz
type fs: float
tau: Time constant in seconds (75us in US, 50us in EUR)
type tau: float</td></tr>
</table>

1.1.9.2) fm_preemph ()

<table>
<tr><td>Type</td><td>Function</td></tr>
<tr><td>Description</td><td>FM Preemphasis IIR filter.

       1 + s*t1
H(s) = ----------
       1 + s*t2
I think this is the right transfer function.

This fine ASCII rendition is based on Figure 5-15
in "Digital and Analog Communication Systems", Leon W. Couch II

<pre>
             R1
        +-----||------+
        | |
 o------+   +-----+--------o
        | C1 | |
        +----/\/\/\/--+   \
                            /
                            \ R2
                            /
                            \
                            |
 o--------------------------+--------o
</pre>
f1 = 1/(2*pi*t1) = 1/(2*pi*R1*C)</td></tr>
</table>

	1 R1 + R2 f2 = ------- = ------------ 2*pi*t2 2*pi*R1*R2*C t1 is 75us in US, 50us in EUR f2 should be higher than our audio bandwidth. The Bode plot looks like this: /---------------- / / <-- slope = 20dB/decade / -------------/ f1 f2 We prewarp and use the bilinear z-transform to get our IIR coefficients. See "Digital Signal Processing: A Practical Approach" by Ifeachor and Jervis
Usage	**blks.fm_deemph(fg, fs, tau=75e-6)**
Parameters	**fg**: flow graph type fg: gr.flow_graph **fs**: sampling frequency in Hz type fs: float **tau**: Time constant in seconds (75us in US, 50us in EUR) type tau: float

1.1.10) gnuradio/blksimpl/gmsk.py

Type	Python file
Description	differential QPSK modulation and demodulation
Examples	See gnuradio-examples/python/digital for examples
Note	

1.1.10.1) gmsk_mod ( )

Type	Function , GMSK modulator
Description	Hierarchical block for Gaussian Minimum Shift Key (GMSK) modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks.gmsk_mod(fg, samples_per_symbol =2, bt=.35, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per baud >= 2 type samples_per_symbol: integer **bt**: Gaussian filter bandwidth * symbol time type bt: float **verbose**: Print information about modulator? type verbose: bool

	debug: Print modulation data to files? type debug: bool

1.1.10.2) gmsk_demod ( )

Type	Function , GMSK demodulator
Description	Hierarchical block for Gaussian Minimum Shift Key (GMSK) demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks.gmsk_demod(fg, samples_per_symbol=2, gain_mu=None, mu=0.5, omega_relative_limit=0.005, freq_error=0.0, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files? type log: bool Clock recovery parameters. These all have reasonable defaults. **gain_mu**: controls rate of mu adjustment type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega type omega_relative_limit: float, typically 0.000200 (200 ppm) **freq_error**: bit rate error as a fraction type freq_error :float

1.1.11) gnuradio/blksimpl/nbfm_rx.py

Type	Python file
Description	Narrow Band FM Receiver
Examples	
Note	

1.1.11.1) nbfm_rx ( )

Type	Function
Description	Narrow Band FM Receiver. Takes a single complex baseband input stream and produces a single float output stream of audio sample in the range [-1, +1].
Usage	**blks.nbfm_rx(fg, audio_rate, quad_rate, tau=75e-6, max_dev=5e3)**
Parameters	**fg**: flow graph **audio_rate**: sample rate of audio stream, >= 16k type audio_rate: integer **quad_rate**: sample rate of output stream, quad_rate must be an integer multiple of audio_rate. type quad_rate: integer **tau**: preemphasis time constant (default 75e-6) type tau: float **max_dev**: maximum deviation in Hz (default 5e3) type max_dev: float

1.1.12) gnuradio/blksimpl/nbfm_tx.py

Type	Python file
Description	Narrow Band FM Transmitter
Examples	
Note	

1.1.12.1) nbfm_tx ( )

Type	Function
Description	Narrow Band FM Transmitter. Takes a single float input stream of audio samples in the range [-1,+1] and produces a single FM modulated complex baseband output.
Usage	**blks.nbfm_tx (fg, audio_rate, quad_rate, tau=75e-6, max_dev=5e3)**
Parameters	**fg**: flow graph **audio_rate**: sample rate of audio stream, >= 16k type audio_rate: integer **quad_rate**: sample rate of output stream, quad_rate must be an integer multiple of audio_rate type quad_rate: integer **tau**: preemphasis time constant (default 75e-6) type tau: float **max_dev**: maximum deviation in Hz (default 5e3) type max_dev: float

1.1.13) gnuradio/blksimpl/ofdm.py

Type	Python file
Description	OFDM mod/demod with packets as i/o
Examples	
Note	

1.1.13.1) ofdm_mod ( )

Type	Function
Description	Modulates an OFDM stream. Based on the options fft_length, occupied_tones, and cp_length, this block creates OFDM symbols using a specified modulation option. Send packets by calling send_pkt Hierarchical block for sending packets. Packets to be sent are enqueued by calling send_pkt. The output is the complex modulated signal at baseband.
Usage	**blks.ofdm_mod (fg, options, msgq_limit=2, pad_for_usrp=True)**
Parameters	**fg**: flow graph type fg: flow graph **options**: pass modulation options from higher layers (fft length, occupied tones, etc.) **msgq_limit**: maximum number of messages in message queue type msgq_limit: int **pad_for_usrp**: If true, packets are padded such that they end up a multiple of 128 samples
Sub Function	Blks.ofdm_mod.send_pkt(payload, eof=False)
Description	Send the payload
Parameters	payload: data to send type payload: string **eof** : To signal end of transmission Type eof : Bool True or False

1.1.13.2) ofdm_demod ( )

Type	Function
Description	Demodulates a received OFDM stream. Based on the options fft_length, occupied_tones, and cp_length, this block performs synchronization, FFT, and demodulation of incoming OFDM symbols and passes packets up the a higher layer. The input is complex baseband. When packets are demodulated, they are passed to the app via the callback. Hierarchical block for demodulating and deframing packets.The input is the complex modulated signal at baseband. Demodulated packets are sent to the handler.
Usage	**blks.ofdm_demod (fg, options, callback=None)**
Parameters	**fg**: flow graph type fg: flow graph **options**: pass modulation options from higher layers (fft length, occupied tones, etc.) **callback**: function of two args: ok, payload type callback: ok: bool; payload: string

1.1.14) gnuradio/blksimpl/ofdm_sync_fixed.py

Type	Python file
Description	OFDM synchronizer
Examples	

1.1.14.1) ofdm_sync_fixed ( )

Type	Function
Description	Use a fixed trigger point instead of sync block
Usage	**blks.ofdm_sync_fixed(fg, fft_length, cp_length, snr)**
Parameters	
Note	Needs more documentaion

1.1.15) gnuradio/blksimpl/ofdm_sync_ml.py

Type	Python file
Description	Maximum Likelihood OFDM synchronizer
Examples	

1.1.15.1) ofdm_sync_ml ( )

Type	Function
Description	Maximum Likelihood OFDM synchronizer: J. van de Beek, M. Sandell, and P. O. Borjesson, "ML Estimation of Time and Frequency Offset in OFDM Systems," IEEE Trans. Signal Processing, vol. 45, no. 7, pp. 1800-1805, 1997.
Usage	**blks.ofdm_sync_ml(fg, fft_length, cp_length, snr, logging)**
Parameters	
Note	Needs more documentation

1.1.16) gnuradio/blksimpl/ofdm_sync_pn.py

Type	Python file
Description	OFDM synchronization using PN Correlation
Examples	

1.1.16.1) ofdm_sync_pn ( )

Type	Function
Description	OFDM synchronization using PN Correlation: T. M. Schmidl and D. C. Cox, "Robust Frequency and Timing Synchonization for OFDM," IEEE Trans. Communications, vol. 45, no. 12, 1997.Signal Processing, vol. 45, no. 7, pp. 1800-1805, 1997.
Usage	**blks.ofdm_sync_pn(fg, fft_length, cp_length, logging=False)**
Parameters	
Note	Needs more documentation

1.1.17) gnuradio/blksimpl/ofdm_sync_pnac.py

Type	Python file
Description	OFDM synchronization using PN Autocorrelation
Examples	

1.1.17.1) ofdm_sync_pnac ( )

Type	Function
Description	OFDM synchronization using Autocorrelation PN
Usage	**blks.ofdm_sync_pnac(fg, fft_length, cp_length, ks)**
Parameters	
Note	Needs more documentation

1.1.18) gnuradio/blksimpl/ofdm_receiver.py

Type	Python file
Description	OFDM Receiver
Examples	
Note	Needs more documentation

1.1.18.1) ofdm_receiver ( )

Type	Function
Description	
Usage	**blks.ofdm_receiver(fg, fft_length, cp_length, occupied_tones, snr, ks, logging=False)**
Parameters	
Note	Needs more documentation

1.1.19) gnuradio/blksimpl/pkt.py

Type	Python file
Description	mod/demod with packets as i/o (sending packets and demodulating /deframing packets)
Examples	
Note	

1.1.19.1) mod_pkts ( )

Type	Function
Description	Wrap an arbitrary digital modulator in our packet handling framework.Send packets by calling send_pkt. Hierarchical block for sending packets.Packets to be sent are enqueued by calling send_pkt.The output is the complex modulated signal at baseband.
Usage	**blks.mod_pkts(fg, modulator, access_code=None, msgq_limit=2, pad_for_usrp=True, use_whitener_offset=False)**
Parameters	**fg**: flow graph type fg: flow graph **modulator**: instance of modulator class (gr_block or hier_block) type modulator: complex baseband out **access_code**: AKA sync vector type access_code: string of 1's and 0's between 1 and 64 long **msgq_limit**: maximum number of messages in message queue type msgq_limit: int **pad_for_usrp**: If true, packets are padded such that they end up a multiple of 128 samples **use_whitener_offset**: If true, start of whitener XOR string is incremented each packet see gmsk_mod for remaining parameters
Sub Function	blks.mod_pkts.send_pkt(payload, eof=False)
Description	Send the payload
Parameters	**payload**: data to send type payload: string **eof** : To signal end of transmission Type eof : Bool True or False

1.1.19.1) demod_pkts ( )

Type	Function
Description	Wrap an arbitrary digital demodulator in our packet handling framework. The input is complex baseband. When packets are demodulated, they are passed to the app via the callback. Hierarchical block for demodulating and deframing packets. The input is the complex modulated signal at baseband. Demodulated packets are sent to the handler.
Usage	**blks.demod_pkts(fg,demodulator,access_code=None,callback=None, threshold=-1)**
Parameters	**fg**: flow graph type fg: flow graph **demodulator**: instance of demodulator class (gr_block or hier_block) type demodulator: complex baseband in **access_code**: AKA sync vector type access_code: string of 1's and 0's **callback**: function of two args: ok, payload type callback: ok: bool; payload: string **threshold**: detect access_code with up to threshold bits wrong (-1 -> use default) type threshold: int

1.1.20) gnuradio/blksimpl/psk.py

Type	Python file
Description	Define different kinds of constellations for Tx and Rx for the PSK (BPSK, QPSK, 8PSK)
Examples	
Note	Needs more Documentation

1.1.21) gnuradio/blksimpl/qam.py

Type	Python file
Description	Define different kinds of constellations for Tx and Rx for the QAM (QAM4,QAM8,QAM16,QAM64,QAM256)
Examples	
Note	Needs more Documentation

1.1.22) gnuradio/blksimpl/qam8.py

Type	Python file
Description	QAM8 modulation and demodulation.
Examples	See gnuradio-examples/python/digital for examples
Note	Needs more Documentation

1.1.22.1) qam8_mod ( )

Type	Function QAM8 modulator
Description	Hierarchical block for RRC-filtered QAM8 modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks.qam8_mod(fg, samples_per_symbol =2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.1.22.2) qam8_demod ( )

Type	Function , QAM8 demodulator
Description	Hierarchical block for QAM8 demodulation. The input is the complex modulated signal at

	baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks.qam8_demod(fg, samples_per_symbol=2, excess_bw=.35, costas_alpha=None, gain_mu=0.03, mu=0.05, omega_relative_limit=0.005, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files? type log: bool **gain_mu**: controls rate of mu adjustment type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega type omega_relative_limit: float, typically 0.000200 (200 ppm)

1.1.23) gnuradio/blksimpl/qam16.py

Type	Python file
Description	QAM16 modulation and demodulation.
Examples	See gnuradio-examples/python/digital for examples
Note	Needs more Documentation

1.1.23.1) qam16_mod ( )

Type	Function QAM16 modulator
Description	Hierarchical block for QAM16 modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks.qam16_mod(fg, samples_per_symbol =2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.1.23.2) qam16_demod ( )

Type	Function , QAM16 demodulator
Description	Hierarchical block for QAM16 demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks.qam16_demod(fg, samples_per_symbol=2, excess_bw=.35, costas_alpha=None, gain_mu=0.03, mu=0.05, omega_relative_limit=0.005, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files? type log: bool **gain_mu**: controls rate of mu adjustment type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega type omega_relative_limit: float, typically 0.000200 (200 ppm)

1.1.24) gnuradio/blksimpl/qam64.py

Type	Python file
Description	QAM64 modulation and demodulation.
Examples	See gnuradio-examples/python/digital for examples
Note	Needs more Documentation

1.1.24.1) qam64_mod ( )

Type	Function QAM64 modulator
Description	Hierarchical block for QAM64 modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks.qam64_mod(fg, samples_per_symbol =2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float

	gray_code: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.1.24.2) qam64_demod ( )

Type	Function , QAM64 demodulator
Description	Hierarchical block for QAM64 demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks.qam64_demod(fg,** **samples_per_symbol=2,** **excess_bw=.35,** **costas_alpha=None,** **gain_mu=0.03,** **mu=0.05,** **omega_relative_limit=0.005,** **gray_code=True,** **verbose=False,** **log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files? type log: bool **gain_mu**: controls rate of mu adjustment type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega type omega_relative_limit: float, typically 0.000200 (200 ppm)

1.1.25) gnuradio/blksimpl/qam256.py

Type	Python file
Description	QAM256 modulation and demodulation.
Examples	See gnuradio-examples/python/digital for examples
Note	Needs more Documentation

1.1.25.1) qam256_mod ( )

Type	Function QAM256 modulator
Description	Hierarchical block for QAM256 modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks.qam256_mod(fg, samples_per_symbol =2,** **excess_bw=.35,** **gray_code=True,** **verbose=False,** **log=False)**
Parameters	**fg**: flow graph

	type fg: flow graph **samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.1.25.2) qam256_demod ( )

Type	Function , QAM256 demodulator
Description	Hierarchical block for QAM256 demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks.qam256_demod(fg, samples_per_symbol=2, excess_bw=.35, costas_alpha=None, gain_mu=0.03, mu=0.05, omega_relative_limit=0.005, gray_code=True, verbose=False, log=False)**
Parameters	**fg**: flow graph type fg: flow graph **samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files? type log: bool **gain_mu**: controls rate of mu adjustment type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega type omega_relative_limit: float, typically 0.000200 (200 ppm)

1.1.26) gnuradio/blksimpl/rational_resampler.py

Type	Python file
Description	Rational resample polyphase FIR filter
Examples	
Note	

1.1.26.1) rational_resampler ( )

Type	Function
Description	**rational_resampler_fff** :Rational resampling polyphase FIR filter with float input, float output and float taps. **rational_resampler_ccf** : Rational resampling polyphase FIR filter with complex input, complex output and float taps.

	rational_resampler_ccc : Rational resampling polyphase FIR filter with complex input, complex output and complex taps.
Usage	**blks.rational_resampler_xxx(fg,interpolation, decimation, taps=None, fractional_bw=None)**
Parameters	**fg**: flow graph **interpolation**: interpolation factor type interpolation: integer > 0 **decimation**: decimation factor type decimation: integer > 0 **taps**: optional filter coefficients see blks.filter_design type taps: sequence **fractional_bw**: fractional bandwidth in (0, 0.5), measured at final freq (use 0.4) type fractional_bw: float
Note	Either taps or fractional_bw may be specified, but not both. If neither is specified, a reasonable default, 0.4, is used as the fractional_bw.

1.1.26.2) design_filter ( )

Type	Function
Description	Given the interpolation rate, decimation rate and a fractional bandwidth, design a set of taps. returns: sequence of numbers
Usage	**blks.design_filter(design_filter(interpolation,decimation, fractional_bw)**
Parameters	**interpolation**: interpolation factor type interpolation: integer > 0 **decimation**: decimation factor type decimation: integer > 0 **fractional_bw**: fractional bandwidth in (0, 0.5) 0.4 works well. type fractional_bw: float

1.1.27) gnuradio/blksimpl/standard_squelch.py

Type	Python file
Description	Implement the squelch function
Examples	
Note	Needs more Documentation

1.1.27.1) standard_squelch ( )

Type	Function
Description	Implement the squelch function with 100msec time constant.
Usage	**blks.standard_squelch(fg, audio_rate)**
Parameters	**fg**: flow graph **audio_rate** : Sample rate of audio stream
Sub Function 1	blks. standard_squelch.set_threshold(threshold)
Description	Set Squelch Threshold value
Sub Function 2	blks. standard_squelch.threshold()
Description	Return Squelch Threshold value
Sub Function 3	blks. standard_squelch.squelch_range()
Description	Return Squelch range

1.1.28) gnuradio/blksimpl/wfm_rcv.py

Type	Python file
Description	Demodulating a broadcast FM signal with a deemphasis

Examples	
Note	

1.1.28.1) wfm_rcv ( )

Type	Function
Description	Hierarchical block for demodulating a broadcast FM signal. The input is the down converted complex baseband signal (gr_complex).The output is the demodulated audio (float).
Usage	**blks.wfm_rcv(fg, quad_rate, audio_decimation)**
Parameters	**fg**: flow graph. type fg: flow graph **quad_rate**: input sample rate of complex baseband input. type quad_rate: float **audio_decimation**: how much to decimate quad_rate to get to audio. type audio_decimation: integer

1.1.29) gnuradio/blksimpl/wfm_rcv_pll.py

Type	Python file
Description	Stereo demodulating a broadcast FM signal with a deemphasis
Examples	
Note	

1.1.29.1) wfm_rcv_pll ( )

Type	Function
Description	Hierarchical block for demodulating a broadcast FM signal. The input is the down converted complex baseband signal (gr_complex).The output is two streams of the demodulated audio (float) 0=Left, 1=Right.
Usage	**blks.wfm_rcv_pll(fg, demod_rate, audio_decimation)**
Parameters	**fg**: flow graph. type fg: flow graph **demod_rate**: input sample rate of complex baseband input. type demod_rate: float **audio_decimation**: how much to decimate demod_rate to get to audio. type audio_decimation: integer

1.1.30) gnuradio/blksimpl/wfm_tx.py

Type	Python file
Description	Wide Band FM Transmitter with a preemphasis
Examples	
Note	

1.1.30.1) wfm_tx ( )

Type	Function
Description	Wide Band FM Transmitter. Takes a single float input stream of audio samples in the range [-1,+1]and produces a single FM modulated complex baseband output.
Usage	**blks. wfm_tx(fg, audio_rate, quad_rate, tau=75e-6, max_dev=75e3)**

Parameters	**fg**: flow graph **audio_rate**: sample rate of audio stream, >= 16k type audio_rate: integer **quad_rate**: sample rate of output stream, quad_rate must be an integer multiple of audio_rate. type quad_rate: integer **tau**: preemphasis time constant (default 75e-6) type tau: float **max_dev**: maximum deviation in Hz (default 75e3) type max_dev: float

1.1.31) gnuradio/blksimpl/cvsd.py

Type	Python file
Description	CVSD encoder and decoder
Examples	
Note	Needs more documentation

1.1.31.1) cvsd_encode ( )

Type	Function
Description	This is a wrapper for the CVSD encoder that performs interpolation and filtering necessary to work with the vocoding. It converts an incoming float (+-1) to a short, scales it (to 32000; slightly below the maximum value), interpolates it, and then vocodes it. The incoming sampling rate can be anything, though, of course, the higher the sampling rate and thehigher the interpolation rate are, the better the sound quality. When using the CVSD vocoder, appropriate sampling rates are from 8k to 64k with resampling rates from 1 to 8. A rate of 8k with a resampling rate of 8 provides a good quality signal.
Usage	**blks2. cvsd_encode(resample=8, bw=0.5)**
Parameters	

1.1.31.2) cvsd_decode ( )

Type	Function
Description	This is a wrapper for the CVSD decoder that performs decimation and filtering necessary to work with the vocoding. It converts an incoming CVSD-encoded short to a float, decodes it to a float, decimates it, and scales it (by 32000; slightly below the maximum value to avoid clipping). The sampling rate can be anything, though, of course, the higher the sampling rate and the higher the interpolation rate are, the better the sound quality. When using the CVSD vocoder, appropriate sampling rates are from 8k to 64k with resampling rates from 1 to 8. A rate of 8k with a resampling rate of 8 provides a good quality signal.
Usage	**blks2. cvsd_decode(resample=8, bw=0.5)**
Parameters	

1.2) gnuradio/blks2 sub package

Type	Folder
Description	Semi-hideous kludge to import everything in the blk2simpl directory into the gnuradio.blks2 namespace. The blocks were implemented using hier_block2.

1.2.1) gnuradio/blks2impl/am_demod.py

Type	Python file
Description	AM demodulation block
Examples	
Note	

1.2.1.1) am_demod_cf ( )

Type	Function
Description	Generalized AM demodulation block with audio filtering. This block demodulates a band-limited, complex down-converted AM channel into the the original baseband signal, applying low pass filtering to the audio output. It produces a float stream in the range [-1.0, +1.0].
Usage	**blks2.am_demod_cf (channel_rate, audio_decim, audio_pass, audio_stop)**
Parameters	**channel_rate**: incoming sample rate of the AM baseband type sample_rate: integer **audio_decim**: input to output decimation rate type audio_decim: integer **audio_pass**: audio low pass filter passband frequency type audio_pass: float **audio_stop**: audio low pass filter stop frequency type audio_stop: float

1.2.1.2) demod_10k0a3e_cf()

Type	Function
Description	AM demodulation block, 10 KHz channel.This block demodulates an AM channel conformant to 10K0A3E emission standards, such as broadcast band AM transmissions.
Usage	**blks2.demod_10k0a3e_cf(channel_rate, audio_decim)**
Parameters	**channel_rate**: incoming sample rate of the AM baseband type sample_rate: integer **audio_decim**: input to output decimation rate type audio_decim: integer

1.2.2) gnuradio/blks2impl/channel_model.py

Type	Python file
Description	Creates a channel model
Examples	
Note	

1.2.2.1) channel_model ( )

Type	Function
Description	Creates a channel model that includes: - AWGN noise power in terms of noise voltage

	- A frequency offset in the channel in ratio - A timing offset ratio to model clock difference (clock rate ratio) (epsilon). It is sample rate difference between tx and rx - Multipath taps
Usage	**blks2.channel_model(noise_voltage=0.0, frequency_offset=0.0, epsilon=1.0, taps=[1.0,0.0])**
Parameters	
Sub Function 1	blks2.channel_model.set_noise_voltage(noise_voltage)
Sub Function 2	blks2.channel_model.set_frequency_offset(frequency_offset)
Sub Function 3	blks2.channel_model.set_taps(taps)

1.2.3) gnuradio/blks2impl/cpm.py

Type	Python file
Description	Continuous Phase modulation.
Examples	See gnuradio-examples/python/digital for examples
Note	

1.2.3.1) cpm_mod ( )

Type	Function
Description	Hierarchical block for Continuous Phase Modulation. The input is a byte stream (unsigned char) representing packed bits and the output is the complex modulated signal at baseband. See Proakis for definition of generic CPM signals: s(t)=exp(j phi(t)) phi(t)= 2 pi h int_0^t f(t') dt' f(t)=sum_k a_k g(t-kT) (normalizing assumption: int_0^infty g(t) dt = 1/2)
Usage	**blks2.cpm_mod(samples_per_symbol=2,bits_per_symbol=1, h_numerator=1, h_denominator=2, cpm_type=0,bt=_def_bt, symbols_per_pulse=1, generic_taps numpy.empty(1), verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per baud >= 2 type samples_per_symbol: integer **bits_per_symbol**: bits per symbol type bits_per_symbol: integer **h_numerator**: numerator of modulation index type h_numerator: integer **h_denominator**: denominator of modulation index (numerator and denominator must be relative primes) type h_denominator: integer **cpm_type**: supported types are: 0=CPFSK, 1=GMSK, 2=RC, 3=GENERAL type cpm_type: integer **bt**: bandwidth symbol time product for GMSK type bt: float **symbols_per_pulse**: shaping pulse duration in symbols type symbols_per_pulse: integer **generic_taps**: define a generic CPM pulse shape (sum = samples_per_symbol/2) type generic_taps: array of floats **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.2.4) gnuradio/blks2impl/d8psk.py

Type	Python file
Description	differential 8PSK modulation and demodulation
Examples	See gnuradio-examples/python/digital for examples
Note	

1.2.4.1) d8psk_mod ( )

Type	Function , D8PSK modulator
Description	Hierarchical block for RRC-filtered QPSK modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks2.d8psk_mod(samples_per_symbol=3, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files! type debug: bool

1.2.4.2) d8psk_demod ( )

Type	Function , D8PSK demodulator
Description	Differentially coherent detection of differentially encoded 8psk. Hierarchical block for RRC-filtered DQPSK demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (LSB)
Usage	**blks2.d8psk_demod(samples_per_symbol=3, excess_bw=.35, costas_alpha=.175, gain_mu=.175, mu=0.5, omega_relative_limit=.005, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: float **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **costas_alpha**: loop filter gain type costas_alpha: float **gain_mu**: for M&M block type gain_mu: float **mu**: for M&M block type mu: float **omega_relative_limit**: for M&M block type omega_relative_limit: float **gray_code**: Tell modulator to Gray code the bits

	type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print demodulation data to files? type debug: bool

1.2.5) gnuradio/blks2impl/dbpsk.py

Type	Python file
Description	Differential BPSK modulation and demodulation
Examples	See gnuradio-examples/python/digital for examples
Note	

1.2.5.1) dbpsk_mod ( )

Type	Function , DBPSK modulator
Description	Hierarchical block for RRC-filtered BPSK modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks2.dbpsk_mod(samples_per_symbol=2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	samples_per_symbol: samples per baud >= 2 type samples_per_symbol: integer excess_bw: Root-raised cosine filter excess bandwidth type excess_bw: float gray_code: Tell modulator to Gray code the bits type gray_code: bool verbose: Print information about modulator? type verbose: bool log: Log modulation data to files? type log: bool

1.2.5.2) dbpsk_demod ( )

Type	Function , DBPSK demodulator
Description	Differentially coherent detection of differentially encoded BPSK. Hierarchical block for RRC-filtered DBPSK demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (LSB).
Usage	**blks2.d8psk_demod(samples_per_symbol=2, excess_bw=.35, costas_alpha=.1, gain_mu=None, mu=0.5, omega_relative_limit=.005, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: float **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **costas_alpha**: loop filter gain type costas_alpha: float

	gain_mu: for M&M block type gain_mu: float **mu**: for M&M block type mu: float **omega_relative_limit**: for M&M block type omega_relative_limit: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print demodulation data to files? type debug: bool

1.2.6) gnuradio/blks2impl/dqpsk.py

Type	Python file
Description	differential QPSK modulation and demodulation
Examples	See gnuradio-examples/python/digital for examples
Note	

1.2.6.1) dqpsk_mod ( )

Type	Function , DQPSK modulator
Description	Hierarchical block for RRC-filtered QPSK modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks2.dqpsk_mod(samples_per_symbol=2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.2.6.2) dqpsk_demod ( )

Type	Function , DQPSK demodulator
Description	Differentially coherent detection of differentially encoded QPSK. Hierarchical block for RRC-filtered DQPSK demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (LSB)
Usage	**blks2.d8psk_demod(samples_per_symbol=2, excess_bw=.35, costas_alpha=.15, gain_mu=None,**

	mu=0.5, **omega_relative_limit=.005,** **gray_code=True,** **verbose=False,** **log=False)**
Parameters	**samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: float **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **costas_alpha**: loop filter gain type costas_alpha: float **gain_mu**: for M&M block type gain_mu: float **mu**: for M&M block type mu: float **omega_relative_limit**: for M&M block type omega_relative_limit: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.2.7) gnuradio/blks2impl/filterbank.py

Type	Python file
Description	Include Both Filter Synthesis and Analysis
Examples	See gnuradio-examples/python/usrp folder
Note	

1.2.7.1) synthesis_filterbank ( )

Type	Function

Description	Uniformly modulated polyphase DFT filter bank: synthesis
	See http://cnx.rice.edu/content/m10424/latest
	Takes M complex streams in, produces single complex stream out that runs at M times the input sample rate
	The channel spacing is equal to the input sample rate.
	The total bandwidth and output sample rate are equal the
	input sample rate * nchannels.
	Output stream to frequency mapping:
	channel zero is at zero frequency.
	
	if mpoints is odd:
	
	Channels with increasing positive frequencies come from
	channels 1 through (N-1)/2.
	Channel (N+1)/2 is the maximum negative frequency, and
	frequency increases through N-1 which is one channel lower
	than the zero frequency.
	
	if mpoints is even:

	Channels with increasing positive frequencies come from channels 1 through (N/2)-1. Channel (N/2) is evenly split between the max positive and negative bins. Channel (N/2)+1 is the maximum negative frequency, and frequency increases through N-1 which is one channel lower than the zero frequency. Channels near the frequency extremes end up getting cut off by subsequent filters and therefore have diminished utility.
Usage	**blks2.synthesis_filterbank (mpoints, taps=None)**
Parameters	**mpoints**: Number of freq bins/interpolation factor/subbands **taps**: Filter taps for subband filter
Example	See ayfabtu.py

1.2.7.2) analysis_filterbank ( )

Type	Function
Description	Uniformly modulated polyphase DFT filter bank: analysis. See http://cnx.rice.edu/content/m10424/latest Takes 1 complex stream in, produces M complex streams out that runs at 1/M times the input sample rate. Same channel to frequency mapping as described in filter synthesis.
Usage	**blks2.analysis_filterbank (mpoints,taps)**
Parameters	**mpoints**: number of freq bins/interpolation factor/subbands **taps**: filter taps for subband filter
Examples	See test_dft_analysis

1.2.8) gnuradio/blks2impl/fm_demod.py

Type	Python file
Description	FM demodulation block
Examples	
Note	

1.2.8.1) fm_demod_cf ( )

Type	Function
Description	Generalized FM demodulation block with deemphasis and audio filtering. This block demodulates a band-limited, complex down-converted FM channel into the original baseband signal, optionally applying deemphasis. Low pass filtering is done on the resultant signal. It produces an output float stream in the range of [-1.0, +1.0].
Usage	**blks2.fm_demod_cf (channel_rate, audio_decim, deviation, audio_pass, audio_stop, gain=1.0, tau=75e-6)**
Parameters	**channel_rate**: incoming sample rate of the FM baseband type sample_rate: integer **deviation**: maximum FM deviation (default = 5000) type deviation: float **audio_decim**: input to output decimation rate type audio_decim: integer **audio_pass**: audio low pass filter passband frequency type audio_pass: float **audio_stop**: audio low pass filter stop frequency type audio_stop: float **gain**: gain applied to audio output (default = 1.0) type gain: float **tau**: deemphasis time constant (default = 75e-6), specify 'None' to prevent deemphasis

1.2.8.2) demod_20k0f3e_cf ()

Type	Function
Description	NBFM demodulation block, 20 KHz channels
Usage	**blks2.demod_20k0f3e_cf(channel_rate, audio_decim)**
Parameters	**sample_rate**: incoming sample rate of the FM baseband type sample_rate: integer **audio_decim**: input to output decimation rate type audio_decim: integer

1.2.8.3) demod_200kf3e_cf ()

Type	Function
Description	WFM demodulation block, mono. This block demodulates a complex, down converted, wideband FM channel conforming to 200KF3E emission standards, outputting floats in the range [-1.0, +1.0].
Usage	**blks2.demod_200kf3e_cf(channel_rate, audio_decim)**
Parameters	**sample_rate**: incoming sample rate of the FM baseband type sample_rate: integer **audio_decim**: input to output decimation rate type audio_decim: integer

1.2.9) gnuradio/blks2impl/fm_emph.py

Type	Python file
Description	FM deemphasis and preemphasis IIR filter
Examples	
Note	

1.2.9.1) fm_deemph ( )

Type	Function
Description	FM Deemphasis IIR filter. $H(s) = \frac{1}{1 + s}$ tau is the RC time constant. critical frequency: w_p = 1/tau We prewarp and use the bilinear z-transform to get our IIR coefficients. See "Digital Signal Processing: A Practical Approach" by Ifeachor and Jervis
Usage	**blks2.fm_deemph(fs, tau=75e-6)**
Parameters	**fs**: sampling frequency in Hz type fs: float **tau**: Time constant in seconds (75us in US, 50us in EUR) type tau: float

1.2.9.2) fm_preemph ()

<table>
<tr><td>Type</td><td>Function</td></tr>
<tr><td>Description</td><td>FM Preemphasis IIR filter.

        1 + s*t1
H(s) = ----------
        1 + s*t2
I think this is the right transfer function.

 This fine ASCII rendition is based on Figure 5-15
 in "Digital and Analog Communication Systems", Leon W. Couch II

              R1
        +-----||------+
        | |
 o------+   +-----+--------o
        | C1 | |
        +----/\/\/\/--+   \
                            /
                            \ R2
                            /
                            \
                            |
 o--------------------------+--------o

 f1 = 1/(2*pi*t1) = 1/(2*pi*R1*C)

        1   R1 + R2
 f2 = ------- = ------------
      2*pi*t2   2*pi*R1*R2*C

 t1 is 75us in US, 50us in EUR
 f2 should be higher than our audio bandwidth.

 The Bode plot looks like this:

                 /----------------
                /
               / <-- slope = 20dB/decade
              /
  -------------/
              f1   f2

 We prewarp and use the bilinear z-transform to get our IIR coefficients.
 See "Digital Signal Processing: A Practical Approach" by Ifeachor and Jervis</td></tr>
<tr><td>Usage</td><td>blks2.fm_deemph(fs, tau=75e-6)</td></tr>
<tr><td>Parameters</td><td>fs: sampling frequency in Hz
type fs: float
tau: Time constant in seconds (75us in US, 50us in EUR)
type tau: float</td></tr>
</table>

1.2.10) gnuradio/blks2impl/gmsk.py

Type	Python file
Description	differential QPSK modulation and demodulation
Examples	See gnuradio-examples/python/digital for examples
Note	

1.2.10.1) gmsk_mod ( )

Type	Function , GMSK modulator
Description	Hierarchical block for Gaussian Minimum Shift Key (GMSK) modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks2.gmsk_mod(samples_per_symbol =2, bt=.35, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per baud >= 2 type samples_per_symbol: integer **bt**: Gaussian filter bandwidth * symbol time type bt: float **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.2.10.2) gmsk_demod ( )

Type	Function , GMSK demodulator
Description	Hierarchical block for Gaussian Minimum Shift Key (GMSK) demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks2.gmsk_demod(samples_per_symbol=2, gain_mu=None, mu=0.5, omega_relative_limit=0.005, freq_error=0.0, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files? type log: bool Clock recovery parameters. These all have reasonable defaults. **gain_mu**: controls rate of mu adjustment type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega

	type omega_relative_limit: float, typically 0.000200 (200 ppm) **freq_error**: bit rate error as a fraction type freq_error :float

1.2.11) gnuradio/blks2impl/nbfm_rx.py

Type	Python file
Description	Narrow Band FM Receiver
Examples	
Note	

1.2.11.1) nbfm_rx ( )

Type	Function
Description	Narrow Band FM Receiver. Takes a single complex baseband input stream and produces a single float output stream of audio sample in the range [-1, +1].
Usage	**blks2.nbfm_rx(audio_rate, quad_rate, tau=75e-6, max_dev=5e3)**
Parameters	**audio_rate**: sample rate of audio stream, >= 16k type audio_rate: integer **quad_rate**: sample rate of output stream, quad_rate must be an integer multiple of audio_rate. type quad_rate: integer **tau**: preemphasis time constant (default 75e-6) type tau: float **max_dev**: maximum deviation in Hz (default 5e3) type max_dev: float

1.2.12) gnuradio/blks2impl/nbfm_tx.py

Type	Python file
Description	Narrow Band FM Transmitter
Examples	
Note	

1.2.12.1) nbfm_tx ( )

Type	Function
Description	Narrow Band FM Transmitter. Takes a single float input stream of audio samples in the range [-1,+1] and produces a single FM modulated complex baseband output.
Usage	**blks2.nbfm_tx (audio_rate, quad_rate, tau=75e-6, max_dev=5e3)**
Parameters	**audio_rate**: sample rate of audio stream, >= 16k type audio_rate: integer **quad_rate**: sample rate of output stream, quad_rate must be an integer multiple of audio_rate. type quad_rate: integer **tau**: preemphasis time constant (default 75e-6) type tau: float **max_dev**: maximum deviation in Hz (default 5e3) type max_dev: float

1.2.13) gnuradio/blks2impl/ofdm.py

Type	Python file
Description	OFDM mod/demod with packets as i/o
Examples	
Note	

1.2.13.1) ofdm_mod ( )

Type	Function
Description	Modulates an OFDM stream. Based on the options fft_length, occupied_tones, and cp_length, this block creates OFDM symbols using a specified modulation option. Send packets by calling send_pkt Hierarchical block for sending packets. Packets to be sent are enqueued by calling send_pkt. The output is the complex modulated signal at baseband.
Usage	**blks2.ofdm_mod (options, msgq_limit=2, pad_for_usrp=True)**
Parameters	**options**: pass modulation options from higher layers (fft length, occupied tones, etc.) **msgq_limit**: maximum number of messages in message queue type msgq_limit: int **pad_for_usrp**: If true, packets are padded such that they end up a multiple of 128 samples
Sub Function	blks.ofdm_mod.send_pkt(payload, eof=False)
Description	Send the payload
Parameters	**payload**: data to send type payload: string **eof** : To signal end of transmission Type eof : Bool True or False

1.2.13.2) ofdm_demod ( )

Type	Function
Description	Demodulates a received OFDM stream. Based on the options fft_length, occupied_tones, and cp_length, this block performs synchronization, FFT, and demodulation of incoming OFDM symbols and passes packets up the a higher layer. The input is complex baseband. When packets are demodulated, they are passed to the app via the callback. Hierarchical block for demodulating and deframing packets. The input is the complex modulated signal at baseband. Demodulated packets are sent to the handler.
Usage	**blks2.ofdm_demod (options, callback=None)**
Parameters	**options**: pass modulation options from higher layers (fft length, occupied tones, etc.) **callback**: function of two args: ok, payload type callback: ok: bool; payload: string

1.2.14) gnuradio/blks2impl/pkt.py

Type	Python file
Description	mod/demod with packets as i/o (sending packets and demodulating /deframing packets)
Examples	
Note	

1.2.14.1) mod_pkts ( )

Type	Function
Description	Wrap an arbitrary digital modulator in our packet handling framework. Send packets by calling send_pkt. Hierarchical block for sending packets. Packets to be sent are enqueued by calling send_pkt. The output is the complex modulated signal at baseband.
Usage	**blks2.mod_pkts(modulator, access_code=None, msgq_limit=2, pad_for_usrp=True, use_whitener_offset=False)**
Parameters	**modulator**: instance of modulator class (gr_block or hier_block) type modulator: complex baseband out **access_code**: AKA sync vector type access_code: string of 1's and 0's between 1 and 64 long **msgq_limit**: maximum number of messages in message queue type msgq_limit: int **pad_for_usrp**: If true, packets are padded such that they end up a multiple of 128 samples **use_whitener_offset**: If true, start of whitener XOR string is incremented each packet see gmsk_mod for remaining parameters
Sub Function	blks.mod_pkts.send_pkt(payload, eof=False)
Description	Send the payload
Parameters	**payload**: data to send type payload: string **eof** : To signal end of transmission Type eof : Bool True or False

1.2.14.2) demod_pkts ( )

Type	Function
Description	Wrap an arbitrary digital demodulator in our packet handling framework. The input is complex baseband. When packets are demodulated, they are passed to the app via the callback. Hierarchical block for demodulating and deframing packets. The input is the complex modulated signal at baseband. Demodulated packets are sent to the handler.
Usage	**blks2.demod_pkts(demodulator,access_code=None,callback=None, threshold=-1)**
Parameters	**demodulator**: instance of demodulator class (gr_block or hier_block) type demodulator: complex baseband in **access_code**: AKA sync vector type access_code: string of 1's and 0's **callback**: function of two args: ok, payload type callback: ok: bool; payload: string **threshold**: detect access_code with up to threshold bits wrong (-1 -> use default) type threshold: int

1.2.15) gnuradio/blks2impl/psk.py

Type	Python file
Description	Define different kinds of constellations for Tx and Rx for the PSK (BPSK, QPSK, 8PSK)
Examples	
Note	Needs more Documentation

1.2.16) gnuradio/blks2impl/qam.py

Type	Python file
Description	Define different kinds of constellations for Tx and Rx for the QAM (QAM4,QAM8,QAM16,QAM64,QAM256)
Examples	
Note	Needs more Documentation

1.2.17) gnuradio/blks2impl/qam8.py

Type	Python file
Description	QAM8 modulation and demodulation.
Examples	See gnuradio-examples/python/digital for examples
Note	Needs more Documentation

1.2.17.1) qam8_mod ( )

Type	Function QAM8 modulator
Description	Hierarchical block for RRC-filtered QAM8 modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks2.qam8_mod(samples_per_symbol =2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.2.17.2) qam8_demod ( )

Type	Function , QAM8 demodulator
Description	Hierarchical block for QAM8 demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks2.qam8_demod(samples_per_symbol=2, excess_bw=.35, costas_alpha=None, gain_mu=0.03, mu=0.05, omega_relative_limit=0.005, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files?

	type log: bool **gain_mu**: controls rate of mu adjustment type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega type omega_relative_limit: float, typically 0.000200 (200 ppm)

1.2.18) gnuradio/blks2impl/qam16.py

Type	Python file
Description	QAM16 modulation and demodulation.
Examples	See gnuradio-examples/python/digital for examples
Note	Needs more Documentation

1.2.18.1) qam16_mod ( )

Type	Function QAM16 modulator
Description	Hierarchical block for QAM16 modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks2.qam16_mod(samples_per_symbol =2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.2.18.2) qam16_demod ( )

Type	Function , QAM16 demodulator
Description	Hierarchical block for QAM16 demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks2.qam16_demod(samples_per_symbol=2, excess_bw=.35, costas_alpha=None, gain_mu=0.03, mu=0.05, omega_relative_limit=0.005, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files? type log: bool **gain_mu**: controls rate of mu adjustment

	type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega type omega_relative_limit: float, typically 0.000200 (200 ppm)

1.2.19) gnuradio/blks2impl/qam64.py

Type	Python file
Description	QAM64 modulation and demodulation.
Examples	See gnuradio-examples/python/digital for examples
Note	Needs more Documentation

1.2.19.1) qam64_mod ( )

Type	Function QAM64 modulator
Description	Hierarchical block for QAM64 modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks2.qam64_mod(samples_per_symbol =2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.2.19.2) qam64_demod ( )

Type	Function , QAM64 demodulator
Description	Hierarchical block for QAM64 demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks2.qam64_demod(samples_per_symbol=2, excess_bw=.35, costas_alpha=None, gain_mu=0.03, mu=0.05, omega_relative_limit=0.005, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files?

	type log: bool **gain_mu**: controls rate of mu adjustment type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega type omega_relative_limit: float, typically 0.000200 (200 ppm)

1.2.20) gnuradio/blks2impl/qam256.py

Type	Python file
Description	QAM256 modulation and demodulation.
Examples	See gnuradio-examples/python/digital for examples
Note	Needs more Documentation

1.2.20.1) qam256_mod ( )

Type	Function QAM256 modulator
Description	Hierarchical block for QAM256 modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	**blks2.qam256_mod(samples_per_symbol =2, excess_bw=.35, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per symbol >= 2 type samples_per_symbol: integer **excess_bw**: Root-raised cosine filter excess bandwidth type excess_bw: float **gray_code**: Tell modulator to Gray code the bits type gray_code: bool **verbose**: Print information about modulator? type verbose: bool **debug**: Print modulation data to files? type debug: bool

1.2.20.2) qam256_demod ( )

Type	Function , QAM256 demodulator
Description	Hierarchical block for QAM256 demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	**blks2.qam256_demod(samples_per_symbol=2, excess_bw=.35, costas_alpha=None, gain_mu=0.03, mu=0.05, omega_relative_limit=0.005, gray_code=True, verbose=False, log=False)**
Parameters	**samples_per_symbol**: samples per baud type samples_per_symbol: integer **verbose**: Print information about modulator? type verbose: bool **log**: Print modulation data to files?

	type log: bool **gain_mu**: controls rate of mu adjustment type gain_mu: float **mu**: fractional delay [0.0, 1.0] type mu: float **omega_relative_limit**: sets max variation in omega type omega_relative_limit: float, typically 0.000200 (200 ppm)

1.2.21) gnuradio/blks2impl/rational_resampler.py

Type	Python file
Description	Rational resample polyphase FIR filter
Examples	
Note	

1.2.21.1) rational_resampler ( )

Type	Function
Description	**rational_resampler_fff** :Rational resampling polyphase FIR filter with float input, float output and float taps. **rational_resampler_ccf** : Rational resampling polyphase FIR filter with complex input, complex output and float taps. **rational_resampler_ccc** : Rational resampling polyphase FIR filter with complex input, complex output and complex taps.
Usage	**blks2.rational_resampler_xxx(interpolation, decimation, taps=None, fractional_bw=None)**
Parameters	**interpolation**: interpolation factor type interpolation: integer > 0 **decimation**: decimation factor type decimation: integer > 0 **taps**: optional filter coefficients see blks2.design_filter type taps: sequence **fractional_bw**: fractional bandwidth in (0, 0.5), measured at final freq (use 0.4) type fractional_bw: float
Note	Either taps or fractional_bw may be specified, but not both. If neither is specified, a reasonable default, 0.4, is used as the fractional_bw.

1.2.21.2) design_filter ( )

Type	Function
Description	Given the interpolation rate, decimation rate and a fractional bandwidth, design a set of taps. returns: sequence of numbers
Usage	**blks2.design_filter(design_filter(interpolation,decimation, fractional_bw)**
Parameters	**interpolation**: interpolation factor type interpolation: integer > 0 **decimation**: decimation factor type decimation: integer > 0 **fractional_bw**: fractional bandwidth in (0, 0.5) 0.4 works well. type fractional_bw: float

1.2.22) gnuradio/blks2impl/standard_squelch.py

Type	Python file
Description	Implement the squelch function
Examples	
Note	Needs more Documentation

1.2.22.1) standard_squelch ( )

Type	Function
Description	Implement the squelch function with 100msec time constant
Usage	**blks2. standard_squelch(audio_rate)**
Parameters	audio_rate : Audio rate
Sub Function 1	blks2. standard_squelch.set_threshold(threshold)
Description	Set Squelch Threshold value
Sub Function 2	blks2. standard_squelch.threshold()
Description	Return Squelch Threshold value
Sub Function 3	blks2. standard_squelch.squelch_range()
Description	Return Squelch range

1.2.23) gnuradio/blks2impl/wfm_rcv.py

Type	Python file
Description	Demodulating a broadcast FM signal with a deemphasis
Examples	
Note	

1.2.23.1) wfm_rcv ( )

Type	Function
Description	Hierarchical block for demodulating a broadcast FM signal. The input is the down converted complex baseband signal (gr_complex).The output is the demodulated audio (float).
Usage	**blks2.wfm_rcv(quad_rate, audio_decimation)**
Parameters	**quad_rate**: input sample rate of complex baseband input. type quad_rate: float **audio_decimation**: how much to decimate quad_rate to get to audio. type audio_decimation: integer

1.2.24) gnuradio/blks2impl/wfm_rcv_pll.py

Type	Python file
Description	Stereo demodulating a broadcast FM signal with a deemphasis
Examples	
Note	

1.2.24.1) wfm_rcv_pll ( )

Type	Function
Description	Hierarchical block for demodulating a broadcast FM signal. The input is the down converted complex baseband signal (gr_complex).The output is two streams of the demodulated audio (float) 0=Left, 1=Right.
Usage	**blks2.wfm_rcv_pll(demod_rate, audio_decimation)**
Parameters	**demod_rate**: input sample rate of complex baseband input. type demod_rate: float **audio_decimation**: how much to decimate demod_rate to get to audio. type audio_decimation: integer

1.2.25) gnuradio/blks2impl/wfm_tx.py

Type	Python file
Description	Wide Band FM Transmitter with a preemphasis
Examples	
Note	

1.2.25.1) wfm_tx ( )

Type	Function
Description	Wide Band FM Transmitter. Takes a single float input stream of audio samples in the range [-1,+1]and produces a single FM modulated complex baseband output.
Usage	**blks2. wfm_tx(audio_rate, quad_rate, tau=75e-6, max_dev=75e3)**
Parameters	**audio_rate**: sample rate of audio stream, >= 16k type audio_rate: integer **quad_rate**: sample rate of output stream, quad_rate must be an integer multiple of audio_rate. type quad_rate: integer **tau**: preemphasis time constant (default 75e-6) type tau: float **max_dev**: maximum deviation in Hz (default 75e3) type max_dev: float

1.3) gnuradio/wxgui sub package

Type	Folder
Description	Wxpython based gnuradio extension

1.3.1) gnuradio/wxgui/fftsink.py

Type	Python file
Description	Gnuradio spectrum analyzer
Examples	
Note	

1.3.1.1) fft_sink_x ( )

Type	Function
Description	FFT sink block. **fft_sink_c ()** : fft sink block for complex data samples.

	fft_sink_f () : fft sink block for real floating data samples.
Usage	**fftsink.fft_sink_x(fg, parent, baseband_freq=0, y_per_div=10, ref_level=50, sample_rate=1, fft_size=512, fft_rate=15, average=False, avg_alpha=None, title='', size=(640,240), peak_hold=False)**
Parameters	

1.3.2) gnuradio/wxgui/fftsink2.py

Type	Python file
Description	Gnuradio spectrum analyzer using stdgui2 and heir_block2
Examples	
Note	

1.3.2.1) fft_sink_x ( )

Type	Function
Description	FFT sink block. **fft_sink_c ()** : fft sink block for complex data samples. **fft_sink_f ()** : fft sink block for real floating data samples.
Usage	**fftsink2.fft_sink_x(parent, baseband_freq=0, y_per_div=10, ref_level=50, sample_rate=1, fft_size=512, fft_rate=15, average=False, avg_alpha=None, title='', size=(640,240), peak_hold=False)**
Parameters	

1.3.3) gnuradio/wxgui/scopesink.py

Type	Python file
Description	Building block for python oscilloscope module.
Examples	
Note	

1.3.3.1) scope_sink_x ( )

Type	Function
Description	Oscilloscope sink block. **scope_sink_c ()** : scope sink block for complex data samples. **scope_sink_f ()** : scope sink block for real floating data samples. Accepts 1 to 16 float streams.
Usage	**scopesink.scope_sink_x(fg, parent, title='', sample_rate=1, size=(640,240), frame_decim=1, v_scale=1000, t_scale=None)**
Parameters	

1.3.4) gnuradio/wxgui/scopesink2.py

Type	Python file
Description	Gnuradio Oscilloscope using stdgui2 and heir_block2
Examples	
Note	

1.3.4.1) scope_sink_x ( )

Type	Function
Description	Scope sink block. **scope_sink_c ()** : scope sink block for complex data samples. **scope_sink_f ()** : scope sink block for real floating data samples. Accepts 1 to 16 float streams.
Usage	**scopesink2.scope_sink_x(parent, title='', sample_rate=1, size=default_scopesink_size, frame_decim=1, v_scale=1000, t_scale=None, num_inputs=1)**
Parameters	

1.3.4.2) constellation_sink ( )

Type	Function
Description	Constellation sink block.
Usage	**scopesink2.constellation_sink(parent,title='Constellation', sample_rate=1, size=(640,240), frame_decim=1)**
Parameters	

1.3.5) gnuradio/wxgui/form.py

Type	Python file
Description	Gnuradio wxgui form
Examples	
Note	

1.3.6) gnuradio/wxgui/numbersink.py

Type	Python file
Description	Gnuradio Number Sink
Examples	
Note	

1.3.6.1) number_sink_x ( )

Type	Function
Description	Number sink block. **number_sink_c ()** : number sink block for complex data samples. **number_sink_f ()** : number sink block for real floating data samples.
Usage	**numbersink.number_sink_x(fg, parent, unit='',base_value=0,minval=-100.0,maxval=100.0,factor=1.0, decimal_places=10, ref_level=50, sample_rate=1, number_rate=15, average=False, avg_alpha=None, label='', size=(640,240), peak_hold=False)**
Parameters	

1.3.7) gnuradio/wxgui/numbersink2.py

Type	Python file
Description	Gnuradio Number Sink using hier_block2

Examples	
Note	

1.3.7.1) number_sink_x ( )

Type	Function
Description	Number sink block. **number_sink_c ()** : number sink block for complex data samples. **number_sink_f ()** : number sink block for real floating data samples.
Usage	**numbersink2.number_sink_x(fg, parent, unit='',base_value=0,minval=-100.0,maxval=100.0,factor=1.0, decimal_places=10, ref_level=50, sample_rate=1, number_rate=15, average=False, avg_alpha=None, label='', size=(640,240), peak_hold=False)**
Parameters	

1.3.8) gnuradio/wxgui/waterfallsink.py

Type	Python file
Description	Gnuradio Waterfall Sink
Examples	
Note	

1.3.8.1) waterfall_sink_x ( )

Type	Function
Description	Waterfall sink block. **waterfall_sink_c ()** : waterfall sink block for complex data samples. **waterfall_sink_f ()** : waterfall sink block for real floating data samples.
Usage	**waterfallsink.number_sink_x(fg, parent, baseband_freq=0, y_per_div=10, ref_level=50, sample_rate=1, fft_size=512, fft_rate=15, average=False, avg_alpha=None, title='', size=(640,240))**
Parameters	

1.3.9) gnuradio/wxgui/waterfallsink2.py

Type	Python file
Description	Gnuradio Waterfall Sink using hier_block2
Examples	
Note	

1.3.9.1) waterfall_sink_x ( )

Type	Function
Description	Waterfall sink block. **waterfall_sink_c ()** : waterfall sink block for complex data samples. **waterfall_sink_f ()** : waterfall sink block for real floating data samples.
Usage	**waterfallsink2.number_sink_x(fg, parent, baseband_freq=0, y_per_div=10, ref_level=50, sample_rate=1, fft_size=512, fft_rate=15, average=False, avg_alpha=None, title='', size=(640,240))**
Parameters	

1.3.10) gnuradio/wxgui/plot.py

Type	Python file
Description	This is a simple light weight plotting module that is used with gnuradio.
Examples	
Note	

1.3.11) gnuradio/wxgui/powermate.py

Type	Python file
Description	Handler for Griffin PowerMate, Contour ShuttlePro & ShuttleXpress USB knobs.This is Linux and wxPython specific
Examples	
Note	Needs more documentation

1.3.12) gnuradio/wxgui/silder.py

Type	Python file
Description	Return a wx.Slider object
Examples	
Note	Needs more documentation

1.3.13) gnuradio/wxgui/stdgui.py

Type	Python file
Description	A simple wx gui for GNU Radio applications
Examples	
Note	Needs more documentation

1.3.14) gnuradio/wxgui/stdgui2.py

Type	Python file
Description	A simple wx gui for GNU Radio applications using hier_block2
Examples	
Note	Needs more documentation

1.3.15) gnuradio/wxgui/ra_fftsink.py

Type	Python file
Description	Radio astronomy gnuradio spectrum analyzer
Examples	
Note	

1.3.15.1) ra_fft_sink_x ( )

Type	Function
Description	FFT sink block. **ra_fft_sink_c ()** : fft sink block for complex data samples.

	ra_fft_sink_f () : fft sink block for real floating data samples.
Usage	**ra_fftsink.ra_fft_sink_x(fg, parent, baseband_freq=0, y_per_div=10, sc_y_per_div=0.5, sc_ref_level=40, ref_level=50, sample_rate=1, fft_size=512, fft_rate=15, average=False, avg_alpha=None, title='', size=(640,140), peak_hold=False, ofunc=None, xydfunc=None)**
Parameters	

1.3.16) gnuradio/wxgui/ra_fftsink.py

Type	Python file
Description	Radio astronomy gnuradio spectrum analyzer
Examples	
Note	

1.3.16.1) ra_fft_sink_x ( )

Type	Function
Description	FFT sink block. **ra_fft_sink_c ()** : fft sink block for complex data samples. **ra_fft_sink_f ()** : fft sink block for real floating data samples.
Usage	**ra_fftsink.ra_fft_sink_x(fg, parent, baseband_freq=0, y_per_div=10, sc_y_per_div=0.5, sc_ref_level=40, ref_level=50, sample_rate=1, fft_size=512, fft_rate=15, average=False, avg_alpha=None, title='', size=(640,140), peak_hold=False, ofunc=None, xydfunc=None)**
Parameters	

1.3.17) gnuradio/wxgui/ra_waterfallsink.py

Type	Python file
Description	Radio Astronomy gnuradio Waterfall Sink
Examples	
Note	

1.3.17.1) waterfall_sink_x ( )

Type	Function
Description	Waterfall sink block. **waterfall_sink_c ()** : waterfall sink block for complex data samples. **waterfall_sink_f ()** : waterfall sink block for real floating data samples.
Usage	**ra_waterfallsink.number_sink_x(fg, parent, baseband_freq=0, ref_level=0, sample_rate=1, fft_size=512, fft_rate=15, average=False, avg_alpha=None, title='', size=(640,240), report=None, span=40, ofunc=None, xydfunc=None)**
Parameters	

1.3.18) gnuradio/wxgui/ra_stripcharsink.py

Type	Python file
Description	Radio Astronomy gnuradio ??????????????????????
Examples	
Note	Needs more documentation

1.3.18.1) stripchar_sink_x ( )

Type	Function
Description	???????????????
Usage	**ra_stripcharsink.stripchart_sink_f(fg, parent, y_per_div=10, ref_level=50, sample_rate=1, title='', stripsize=4, size=(640,140),xlabel="X", ylabel="Y", divbase=0.025, parallel=False, scaling=1.0, autoscale=False)**
Parameters	

1.4) gnuradio/vocoder sub package

Type	Folder
Description	GSM and CVSD blocks

1.4.1) gnuradio/vocoder/gsm_full_rate.py

Type	Python file
Description	GSM 6.10 Full rate encoder decoder blocks
Examples	
Note	Needs more documentation

1.4.1.1) encode_sp ( )

Type	Function
Description	GSM 06.10 Full Rate Vocoder Encoder block input type is short, output type packets.
Usage	**gsm_full_rate.encode_sp()**
Parameters	shorts in 33 byte packets out

1.4.1.2) decode_ps ( )

Type	Function
Description	GSM 06.10 Full Rate Vocoder Decoder block input type is packets, output type short.
Usage	**gsm_full_rate.decode_ps()**
Parameters	33 byte packets in shorts out

1.4.2) gnuradio/vocoder/cvsd_vocoder.py

Type	Python file
Description	CVSD encoder decoder blocks
Examples	
Note	Needsmore documentation

1.4.2.1) encode_sb ( )

Type	Function
Description	This block performs CVSD audio encoding. Its design and implementation is modeled after the CVSD encoder/decoder specifications defined in the Bluetooth standard. CVSD Vocoder Encoder block input type is shorts, output type bytes.
Usage	**cvsd_vocoder.encode_sb()**
Parameters	shorts in bytes out

1.4.2.2) decode_bs ( )

Type	Function
Description	This block performs CVSD audio decoding. Its design and implementation is modeled after the CVSD encoder/decoder specifications defined in the Bluetooth standard CVSD Vocoder Decoder block input type is bytes, output type shorts.
Usage	**cvsd_vocoder.decode_bs()**
Parameters	Bytes in shorts out

1.5) gnuradio/pager sub package

Type	Folder
Description	Radio Pager receiver blocks

1.5.1) gnuradio/pager/flex_demod.py

Type	Python file
Description	This GNU Radio component implements a FLEX radio pager receiver/demodulator. FLEX pager towers are between 929 MHz and 932 MHz at 25 KHz centers. Current status (7/16/07): FLEX receiving is completed except for addition of BCH error correction.
Examples	See /gnuradio/gr-pager
Note	Needs more documentaion

1.5.1.1) flex_demod ( )

Type	Function
Description	FLEX pager protocol demodulation block. This block demodulates a band-limited, complex down-converted baseband channel into FLEX protocol frames.
Usage	**pager.flex_demod(queue, freq=0.0, verbose=False, log=False)**
Parameters	

1.5.2) gnuradio/pager/pager_swig.py

Type	Python file
Description	This file was automatically generated by SWIG. It contains all the necessary software components to implement pager flex demodulation block.

Examples	
Note	Needs more documentation
Sub Function 1	pager_flex_deinterleave()
Sub Function 2	pager_flex_frame()
Sub Function 3	pager_flex_parse()
Sub Function 4	pager_flex_sync()
Sub Function 5	pager_slicer_fb()

1.5.3) gnuradio/pager/aypabtu.py

Type	Python file
Description	All your pager applications belong to us. This is a general program that uses the USRP to demodulate any pager bandwidth. Program options are : --upper-freq", type="eng_float", help="lower Rx frequency --lower-freq", type="eng_float", help="upper Rx frequency --rx-board", type="subdev", help="select USRP Rx side A or B (default=first daughterboard found)" --calibration", type="eng_float", default=0.0, help="set frequency offset to Hz" --gain", type="int", help="set RF gain"
Examples	See gnuradio/gr-pager/README
Note	

1.5.4) gnuradio/pager/usrp_flex.py

Type	Python file
Description	This example application demonstrates receiving and demodulating the FLEX pager protocol.
Examples	See gnuradio/gr-pager/README
Note	

1.5.5) gnuradio/pager/usrp_flex_all.py

Type	Python file
Description	This application demonstrates receiving and demodulating the FLEX pager protocol for the entire 3 MHz band.
Examples	See gnuradio/gr-pager/README
Note	

1.5.6) gnuradio/pager/usrp_flex_band.py

Type	Python file
Description	This application demonstrates receiving and demodulating the FLEX pager protocol for 1 MHz band.
Examples	See gnuradio/gr-pager/README
Note	

1.6) gnuradio/gruimpl sub package

Type	Folder
Description	Gnuradio Utility implementation package

1.6.1) gnuradio/ gruimpl /crc.py

Type	Python file
Description	This GNU Radio component implements CRC generation and checking
Examples	
Note	Needs more documentation

1.6.1.1) gen_and_append_crc32 ( )

Type	Function
Description	Generate CRC
Usage	**gru.gen_and_append_crc32(s)**
Parameters	**s**: String

1.6.1.2) check_crc32 ( )

Type	Function
Description	Generate CRC
Usage	**gru.check_crc32(s)**
Parameters	**s**: String

1.6.2) gnuradio/ gruimpl /freqz.py

Type	Python file
Description	Compute frequency response of a digital filter
Examples	
Note	Needs more documentation

1.6.2.1) freqz ( )

Type	Function
Description	Given the numerator (b) and denominator (a) of a digital filter compute its frequency response. $H(e^{jw}) = \frac{B(e^{jw})}{A(e^{jw})} = \frac{b[0] + b[1]e^{-jw} + ... + b[m]e^{-jmw}}{a[0] + a[2]e^{-jw} + ... + a[n]e^{-jnw}}$ Inputs: b, a --- the numerator and denominator of a linear filter. worN --- If None, then compute at 512 frequencies around the unit circle. If a single integer, the compute at that many frequencies. Otherwise, compute the response at frequencies given in worN whole -- Normally, frequencies are computed from 0 to pi (upper-half of unit-circle. If whole is non-zero compute frequencies from 0 to 2*pi.

	Outputs: (h,w) h -- The frequency response. w -- The frequencies at which h was computed.
Usage	**gru.freqz(b, a, worN=None, whole=0, plot=None)**
Parameters	**b, a** : The numerator and denominator of a linear filter. **worN** : If None, then compute at 512 frequencies around the unit circle. If a single integer, the compute at that many frequencies. Otherwise, compute the response at frequencies given in worn **whole** : Normally, frequencies are computed from 0 to pi (upper-half of unit-circle. If whole is non-zero compute frequencies from 0 to 2*pi.

1.6.3) gnuradio/ gruimpl /gnuplot_freqz.py

Type	Python file
Description	Plot the frequency response of a digital filter using Gnuplot
Usage	**gru.gnuplot_freqz(taps, sample_rate)**
Parameters	**taps** : taps generated by gru.freqz **sample_rate** : ??????
Examples	See ayfabtu.py
Note	Needs more documentation

1.6.3.1) gnuplot_freqz ( )

Type	Function
Description	Plot the frequency response of a digital filter using Gnuplot. Returns a handle to the gnuplot graph. When the handle is reclaimed the graph is torn down
Usage	**gru.gnuplot_freqz (hw, Fs=None, logfreq=False)**
Parameters	**hw** : is a tuple of the form (h, w) where h is sequence of complex freq responses, and w is a sequence of corresponding frequency points. Plot the frequency response using gnuplot. **Fs** : If Fs is provided, use it as the sampling frequency, else use 2*pi.

1.6.4) gnuradio/ gruimpl /gnuplot_freqz.py

Type	Python file
Description	Plot the frequency response of a digital filter using Gnuplot
Examples	
Note	Needs more documentation

1.6.4.1) gnuplot_freqz ( )

Type	Function
Description	Plot the frequency response of a digital filter using gnuplot. Returns a handle to the gnuplot graph. When the handle is reclaimed the graph is torn down
Usage	**gru.gnuplot_freqz (hw, Fs=None, logfreq=False)**
Parameters	**hw** : is a tuple of the form (h, w) where h is sequence of complex freq responses, and w is a sequence of corresponding frequency points. Plot the frequency response using gnuplot. **Fs** : If Fs is provided, use it as the sampling frequency, else use 2*pi.

1.6.5) gnuradio/ gruimpl /hexint.py

Type	Python file
Description	Convert unsigned masks into signed integers
Examples	
Note	

1.6.5.1) hexint ( )

Type	Function
Description	Convert unsigned masks into signed integets. This allows us to use hex constants like 0xf0f0f0f2 when talking to our hardware and not get screwed by them getting treated as python longs.
Usage	**gru.hexint(mask)**
Parameters	**mask** : hex string

1.6.6) gnuradio/ gruimpl /listmsc.py

Type	Python file
Description	Return a copy of x that is reverse order
Examples	
Note	

1.6.6.1) list_revers ( )

Type	Function
Description	Return a copy of x that is reverse order
Usage	**gru.list_revers(x)**
Parameters	**x** : list

1.6.7) gnuradio/ gruimpl /lmx2306.py

Type	Python file
Description	'''Control National LMX2306 based frequency synthesizer using PC Parallel port
Examples	
Note	

1.6.7.1) lmx2306 ( )

Type	Function
Description	Control the National LMX2306 PLL
Usage	**gru.lmx2306(fosc, step_size, which_pp = 0)**
Parameters	**fosc** : is the frequency of the reference oscillator, **step_size** : is the step between valid frequencies, **which_pp** : specifies which parallel port to use

1.6.8) gnuradio/ gruimpl /mathmisc.py

Type	Python file
Description	Some Math functions like GCD, LCM, Log2
Examples	
Note	
Sub Function 1	**gru.gcd(a.b)**
Sub Function 2	**gru.lcm(a.b)**
Sub Function 3	**gru.log2(x)**

1.6.9) gnuradio/ gruimpl /os_read_exactly.py

Type	Python file
Description	Replacement for os.read that blocks until it reads exactly nbytes.
Examples	
Note	

1.6.9.1) os_read_exactly ( )

Type	Function
Description	Replacement for os.read that blocks until it reads exactly nbytes.
Usage	**gru.os_read_exactly(file_descriptor, nbytes)**
Parameters	

1.6.10) gnuradio/ gruimpl /sdr_1000.py

Type	Python file
Description	Control the DDS on the SDR-1000
Examples	
Note	

1.6.10.1) sdr_1000 ( )

Type	Function
Description	Control the DDS on the SDR-1000
Usage	**gru.sdr_1000(pport=0)**
Parameters	
Sub Function 1	gru.sdr_1000.write_reg(addr, data)
Sub Function 2	gru.sdr_1000.set_freq(freq)
Sub Function 3	gru.sdr_1000.set_band(freq)
Sub Function 4	gru.sdr_1000.set_bit (reg, bit, state)
Sub Function 5	gru.sdr_1000.set_tx(on=1)
Sub Function 6	gru.sdr_1000.set_rx()
Sub Function 7	gru.sdr_1000.set_gain (high)
Sub Function 8	gru.sdr_1000.set_mute (mute = 1)
Sub Function 9	gru.sdr_1000.set_unmute ()
Sub Function 10	gru.sdr_1000.set_external_pin (pin, on = 1)

1.6.11) gnuradio/ gruimpl /seq_with_cursor.py

Type	Python file
Description	?????????????, I think it is like a loopup table item selector. Return a list item indexed by cursor
Usage	**gru.seq_with_cursor(list_array, cursor)**
Parameters	**list_array** : list holds data ??????????? **cursor** : list index ???????
Examples	
Note	Needs more documentation

1.6.12) gnuradio/ gruimpl /socket_stuff.py

Type	Python file
Description	Setup sockets for TCP/UDP connections
Examples	
Note	

1.6.12.1) tcp_connect_or_die ( )

Type	Function
Description	Setup sockets for TCP connections. returns: socket or exits
Usage	**gru.tcp_connect_or_die(sock_addr)**
Parameters	**sock_addr**: (host, port) to connect to type sock_addr: tuple

1.6.12.2) udp_connect_or_die ( )

Type	Function
Description	Setup sockets for UDP connections. returns: socket or exits
Usage	**gru.udp_connect_or_die(sock_addr)**
Parameters	**sock_addr**: (host, port) to connect to type sock_addr: tuple

1.7) gnuradio/gru sub package

Type	Folder
Description	Semi-hideous kludge to import everything in the gruimpl directory into the gnuradio.gru namespace.

1.8) gnuradio/gr sub package

Type	Folder
Description	This is the main GNU Radio python module. We pull the swig output and the other modules into the gnuradio.gr namespace

1.8.1) gnuradio/ gr / basic_flow_graph.py

Type	Python file
Description	Constructs the basic flow graph and provides basic operations on the graph.
Examples	
Note	
Sub Function 1	**connect ()** : Connect blocks. connect requires two or more arguments that can be coerced to endpoints
Sub Function 2	**disconnect ()** : Disconnect blocks. disconnect requires two arguments
Sub Function 3	**disconnect_all()** : disconnect all graph blocks.

1.8.2) gnuradio/ gr / flow_graph.py

Type	Python file
Description	Add physical connection info to basic_flow_graph and play
Examples	
Note	
Sub Function 1	**start()** : Start graph, forking thread(s), return immediately
Sub Function 2	**stop()** : Tells scheduler to stop and waits for it to happen
Sub Function 3	**wait()** : Waits for scheduler to stop.
Sub Function 4	**run():** Start graph, wait for completion
Sub Function 5	**is_running()** : Check if the graph is still running

1.8.3) gnuradio/ gr / exceptions.py

Type	Python file
Description	Exception handling
Examples	
Note	
Sub Function 1	**NotDAG (Exception):**Not a directed acyclic graph
Sub Function 2	**CantHappen (Exception):**Can't happen

1.8.4) gnuradio/ gr / gnuradio_swig_py_filter.py

Type	Python file
Description	This file was automatically generated by SWIG. All digital IIR and FIR filter blocks implemented here.
Examples	
Note	

1.8.4.1) iir_filter_ffd ( )

Type	Function
Description	IIR filter with float input, float output and double taps. This filter uses the Direct Form I implementation, where fftaps contains the feed-forward taps, and fbtaps the feedback ones.
Usage	**gr.iir_filter_ffd(fftaps,fbtaps)**
Parameters	**Fftaps :** contains the feed-forward taps

	fbtaps : the feedback taps
Sub Function 1	gr.iir_filter_ffd.set_taps(fftaps,fbtaps)
Example	See hfx2.py in apps

1.8.4.2) single_pole_iir_filter_xx ( )

Type	Function
Description	Used to do averaging for input vector **single_pole_iir_filter_ff**: single pole IIR filter with float input, float output **single_pole_iir_filter_cc**: single pole IIR filter with complex input, complex output. When alpha =1, no averaging is done. The input and output satisfy a difference equation of the form : y(n)=alpha* x(n)+(1-alpha)y(n-1)
Usage	**gr. single_pole_iir_filter_xx(alpha,vlen)**
Parameters	**alpha** : double , time costant. **vlen** : unsigned integer, vector length
Sub Function 1	gr. single_pole_iir_filter_xx.set_taps(alpha)

1.8.4.3) hilbert_fc ( )

Type	Function
Description	Hilbert transformer FIR filter. Real output is input appropriately delayed. Imaginary output is hilbert filtered (90 degree phase shift) version of input.
Usage	**gr. hilbert_fc(ntaps)**
Parameters	**ntaps** : unsigned integer, number of taps (odd)

1.8.4.4) filter_delay_fc ( )

Type	Function
Description	Filter-Delay Combination Block. The block takes one or two float stream and outputs a complex stream. If only one float stream is input, the real output is a delayed version of this input and the imaginary output is the filtered output. If two floats are connected to the input, then the real output is the delayed version of the first input, and the imaginary output is the filtered output. The delay in the real path accounts for the group delay introduced by the filter in the imaginary path. The filter taps needs to be calculated before initializing this block.
Usage	**gr. filter_delay_fc (taps)**
Parameters	**taps** : vector of float taps

1.8.4.5) fft _filter_xx ( )

Type	Function
Description	**fft_filter_ccc**: Fast FFT filter with complex input, complex output and complex taps. **fft_filter_fff**: Fast FFT filter with float input, float output and float taps
Usage	**gr. fft_filter _xx(decimation, taps)**
Parameters	**decimation** : integer **taps** : float
Sub Function 1	gr. fft_filter _xx.set_taps(taps)

1.8.4.6) fractional_interpolator_xx ( )

Type	Function
Description	**fractional_interpolator**_cc : Interpolating mmse filter with complex input, complex

	output.. **fractional_interpolator_ff** : Interpolating mmse filter with float input, float output.
Usage	**gr. fractional_interpolator(phase_shift,inter_ratio)**
Parameters	**phase_shift** :float **inter_ratio** : float
Sub Function 1	gr. fractional_interpolator_xx.mu() : return mu (phase shift) as a float number
Sub Function 2	gr. fractional_interpolator_xx.inter_ratio() : return interpolation ratio as a float number
Sub Function 3	gr. fractional_interpolator_xx.set_mu(mu) : set float mu (phase shift)
Sub Function 4	gr. fractional_interpolator_xx.set_inter_ratio(inter_ratio) : set float interpolation ratio

1.8.4.7) goertzel_fc ( )

Type	Function
Description	Do the Goertzel single-bin DFT calculation.
Usage	**gr. goertzel_fc (rate, len, freq)**
Parameters	**rate** :integer **len** :integer **freq**: float

1.8.4.8) cma_equalizer_cc ( )

Type	Function
Description	Implements constant modulus adaptive filter on complex stream
Usage	**gr. cma_equalizer_cc(num_taps, modulus,mu)**
Parameters	**num_taps** :integer **modulus**: float **mu**: float (phase shift)

1.8.4.9) adaptive_fir_ccf ( )

Type	Function
Description	Adaptive FIR filter with gr_complex input, gr_complex output and float taps.
Usage	**gr. adaptive_fir_ccf (name, decimation, taps)**
Parameters	**name** : string **decimation** :integer **taps** :list of float
Sub Function 1	gr. adaptive_fir_ccf.set_taps(taps): set a float filter taps
Note	Needs more documentation

1.8.4.10) fir_filter_xxx ( )

Type	Function
Description	**fir_filter_ccc**: FIR filter with gr_complex input, gr_complex output and gr_complex taps **fir_filter_ccf**: FIR filter with gr_complex input, gr_complex output and float taps **fir_filter_fcc**: FIR filter with float input, gr_complex output and gr_complex taps **fir_filter_fff**: FIR filter with float input, float output and float taps **fir_filter_fsf**: FIR filter with float input, short output and float taps **fir_filter_scc**: FIR filter with short input, gr_complex output and gr_complex taps
Usage	**gr. fir_filter_xxx (decimation, taps)**
Parameters	**decimation** :integer **taps**: depends on function

Sub Function 1	gr.fir_filter_xxx.set_taps(taps): set filter taps
Note	Needs more documentation

1.8.4.11) freq_xlating_fir_filter_xxx ( )

Type	Function
Description	Software frequency (DDC or DUC) translation filter. This class efficiently combines a frequency translation (typically "down conversion") with a FIR filter (typically low-pass) and decimation. It is ideally suited for a "channel selection filter" and can be efficiently used to select and decimate a narrow band signal out of wide bandwidth input. Uses a single input array to produce a single output array. Additional inputs and/or outputs are ignored. **freq_xlating_fir_filter_ccc**: FIR filter combined with frequency translation with gr_complex input, gr_complex output and gr_complex taps **freq_xlating_fir_filter _ccf**: FIR filter combined with frequency translation with gr_complex input, gr_complex output and float taps **freq_xlating_fir_filter _fcc**: FIR filter combined with frequency translation with float input, gr_complex output and gr_complex taps **freq_xlating_fir_filter _fcf**: FIR filter combined with frequency translation with float input, complex output and float taps **freq_xlating_fir_filter _scf**: FIR filter combined with frequency translation with short input, complex output and float taps **freq_xlating_fir_filter _scc**: FIR filter combined with frequency translation with short input, gr_complex output and gr_complex taps
Usage	**gr. freq_xlating_fir_filter _xxx (decimation, taps, center_freq, sampling_freq)**
Parameters	**decimation** :integer **taps**: depends on function **center_freq** : double **sampling_freq** : double
Sub Function 1	gr. freq_xlating_fir_filter_xxx.set_taps(taps): set filter taps
Sub Function 2	gr. freq_xlating_fir_filter_xxx.set_center_freq(center_freq): set (type double) center_frequency

1.8.4.12) interp_fir_filter_xxx ( )

Type	Function
Description	**interp_fir_filter_ccc**: Interpolating FIR filter with gr_complex input, gr_complex output and gr_complex taps **interp_fir_filter_ccf**: Interpolating FIR filter with gr_complex input, gr_complex output and float taps **interp_fir_filter_fcc**: Interpolating FIR filter with float input, gr_complex output and gr_complex taps **interp_fir_filter_fff**: Interpolating FIR filter with float input, float output and float taps **interp_fir_filter_fsf**: Interpolating FIR filter with float input, short output and float taps **interp_fir_filter_scc**: Interpolating FIR filter with short input, gr_complex output and gr_complex taps
Usage	**gr. interp_fir_filter _xxx (interpolation, taps)**
Parameters	**interpolation** :integer **taps**: depends on function
Sub Function 1	gr. interp_fir_filter _xxx.set_taps(taps): set filter taps
Note	

1.8.4.13) rational_resampler_base_xxx ( )

Type	Function
Description	**rational_resampler_base_ccc**: Rational Resampling Polyphase FIR filter with gr_complex input, gr_complex output and gr_complex taps **rational_resampler_base_ccf**: Rational Resampling Polyphase FIR filter with gr_complex input, gr_complex output and float taps **rational_resampler_base_fcc**: Rational Resampling Polyphase FIR filter with float input, gr_complex output and gr_complex taps **rational_resampler_base_fff**: Rational Resampling Polyphase FIR filter with float input, float output and float taps **rational_resampler_base_fsf**: Rational Resampling Polyphase FIR filter with float input, short output and float taps **rational_resampler_base_scc**: Rational Resampling Polyphase FIR filter with short input, gr_complex output and gr_complex taps
Usage	**gr. rational_resampler_base _xxx (interpolation, decimation, taps)**
Parameters	**interpolation** :unsigned **decimation** : unsigned **taps**: depends on function
Sub Function 1	gr rational_resampler_base _xxx.set_taps(taps): set filter taps
Sub Function 2	gr rational_resampler_base _xxx.intrpolation(): return interpolation value
Sub Function 1	gr rational_resampler_base _xxx.decimation(): return decimation value
Note	

1.8.5) gnuradio/ gr / gnuradio_swig_py_gengen.py

Type	Python file
Description	This file was automatically generated by SWIG. Some mathematical, source and sink blocks are defined here.
Examples	
Note	

1.8.5.1) add_xx ( )

Type	Function
Description	**add_cc** : output = sum (input_0, input_1, ...). Add across all input complex streams. **add_ii** : output = sum (input_0, input_1, ...). Add across all input integer streams. **add_ss** : output = sum (input_0, input_1, ...). Add across all input short streams. **add_ff** : output = sum (input_0, input_1, ...). Add across all input float streams.
Usage	**gr.add _xx ()**
Parameters	
Note	

1.8.5.2) add_vxx ( )

Type	Function
Description	**add_vcc** : output = sum (input_0, input_1, ...). Add across all input complex vectors. **add_vii** : output = sum (input_0, input_1, ...). Add across all input integer vectors. **add_vff** : output = sum (input_0, input_1, ...). Add across all input float vectors. **add_vss** : output = sum (input_0, input_1, ...). Add across all input short vectors.
Usage	**gr.add _vxx()**
Parameters	
Note	

1.8.5.3) add_const_xx ( )

Type	Function
Description	**add_const_cc** : output = input + complex constant. Add constant to input complex streams. **add_ const_ii** : output = input + integer constant. Add constant to input integer streams. **add_ const_ss** : output = input + short constant. Add constant to input short streams. **add_ const_ff** : output = input + float constant. Add constant to input float streams. **add_ const_sf** : output = input + float constant. Add constant to input short streams.
Usage	**gr.add_const _xx (k)**
Parameters	**k**: constant value, type depends on the function type
Note	
Sub Function 1	gr.add_const_xx.set_k(k): set the constant value on the fly.

1.8.5.4) add_const_vxx ( )

Type	Function
Description	**add_const_vcc** : output vector = input complex vector + constant complex vector. **add_ const_vii** : output vector = input integer vector + constant integer vector. **add_ const_vss** : output vector = input short vector + constant short vector. **add_ const_vff** : output vector = input float vector + constant float vector.
Usage	**gr.add_const _vxx (k)**
Parameters	k: constant value, type depends on the function type
Note	
Sub Function 1	gr.add_const_vxx.set_k (k): set the constant value on the fly.
Sub Function 2	gr.add_const_vxx.k(): return the constant vector

1.8.5.5) argmax_xx ( )

Type	Function
Description	**argmax_fs** : ???????????????? **argmax_is** : ???????????????? **argmax_ss**: ????????????????
Usage	**gr.argmax _xx (vlen)**
Parameters	**vlen** : vector length
Note	Needs more documentation

1.8.5.6) chunks_to_symbols _xx( )

Type	Function
Description	Map a stream of symbol indexes (unpacked bytes or shorts) to stream of float or complex onstellation points.in D dimensions (D = 1 by default). out[n D + k] = symbol_table[in[n] D + k], k=0,1,...,D-1 The combination of gr_packed_to_unpacked_XX followed by gr_chunks_to_symbols_XY handles the general case of mapping from a stream of bytes or shorts into arbitrary float or complex symbols. **chunks_to_symbols_bf** : input: stream of unsigned char; output: stream of float **chunks_to_symbols_bc** : input: stream of unsigned char; output: stream of gr_complex **chunks_to_symbols_sf** : input: stream of shorts; output: stream of float **chunks_to_symbols_sc** : input: stream of shorts; output: stream of gr_complex **chunks_to_symbols_if** : input: stream of integers; output: stream of float **chunks_to_symbols_ic** : input: stream of integers; output: stream of gr_complex
Usage	**gr.chunks_to_symbols _xx(symbol_table, D)**

Parameters	**symbol_table** : ???????????? **D** : dimensions, const integer
Note	
Sub Function 1	gr. chunks_to_symbols_xx.symbol_table(): return symbol table
Sub Function 2	gr. chunks_to_symbols_xx.D(): return dimension

1.8.5.7) packed_to_unpacked _xx( )

Type	Function
Description	Convert a stream of packed bytes or shorts to stream of unpacked bytes or shorts. This is the inverse of gr_unpacked_to_packed_XX. The bits in the bytes or shorts input stream are grouped into chunks of bits_per_chunk bits and each resulting chunk is written right- justified to the output stream of bytes or shorts. All b or 16 bits of the each input bytes or short are processed. The right thing is done if bits_per_chunk is not a power of two. The combination of gr_packed_to_unpacked_XX_ followed by gr_chunks_to_symbols_Xf or gr_chunks_to_symbols_Xc handles the general case of mapping from a stream of bytes or shorts into arbitrary float or complex symbols. **packed_to_unpacked _bb**: input: stream of unsigned char; output: stream of unsigned char **packed_to_unpacked _ii**: input: stream of integers; output: stream of intergers **packed_to_unpacked _ss**: input: stream of shorts; output: stream of shorts
Usage	**gr. packed_to_unpacked _xx(bits_per_chunk, endianness)**
Parameters	**bits_per_chunk** :unsigned int **endianness** : GR_MSB_FIRST, GR_LSB_FIRST
Note	

1.8.5.8) unpacked_to_packed _xx( )

Type	Function
Description	Convert a stream of unpacked bytes or shorts into a stream of packed bytes or shorts. This is the inverse of gr_packed_to_unpacked_XX. The low bits_per_chunk bits are extracted from each input byte or short. These bits are then packed densely into the output bytes or shorts, such that all 8 or 16 bits of the output bytes or shorts are filled with valid input bits. The right thing is done if bits_per_chunk is not a power of two. The combination of gr_packed_to_unpacked_XX followed by gr_chunks_to_symbols_Xf or gr_chunks_to_symbols_Xc handles the general case of mapping from a stream of bytes or shorts into arbitrary float or complex symbols. **unpacked_to_packed _bb**: input: stream of unsigned char; output: stream of unsigned char **unpacked_to_packed _ii**: input: stream of integers; output: stream of intergers **unpacked_to_packed _ss**: input: stream of shorts; output: stream of shorts
Usage	**gr. unpacked_to_packed _xx(bits_per_chunk, endianness)**
Parameters	**bits_per_chunk** :unsigned int **endianness** : GR_MSB_FIRST, GR_LSB_FIRST
Note	

1.8.5.9) divide_xx( )

Type	Function
Description	**divide_cc** : output = input_0 / input_1 / input_x ...) .Divide across all input complex streams. **divide_ss** : output = input_0 / input_1 / input_x ...) .Divide across all input short streams. **divide_ii** : output = input_0 / input_1 / input_x ...) .Divide across all input integer streams. **divide_ff** : output = input_0 / input_1 / input_x ...) .Divide across all input float streams.
Usage	**gr.divide_xx()**
Parameters	
Note	

1.8.5.10) max_xx ( )

Type	Function
Description	**max_ff** : ???????????????? **max_ii** : ???????????????? **max_ss**: ????????????????
Usage	**gr.max _xx (vlen)**
Parameters	**vlen** : Vecter length
Note	Needs more documentation

1.8.5.11) multiply_xx ( )

Type	Function
Description	**multiply_cc** : output = prod (input_0, input_1, ...). Multiply across all input complex streams. **multiply _ii** : output = prod (input_0, input_1, ...). Multiply across all input integer streams. **multiply _ss** : output = prod (input_0, input_1, ...). Multiply across all input short streams. **multiply _ff** : output = prod (input_0, input_1, ...). Multiply across all input float streams.
Usage	**gr.multiply _xx ()**
Parameters	
Note	

1.8.5.12) multiply_vxx ( )

Type	Function
Description	**multiply_vcc** : output = prod (input_0, input_1, ...). Element-wise multiply across all input complex vectors **multiply _vii** : output = prod (input_0, input_1, ...). Element-wise multiply across all input integer vectors **multiply _vss** : output = prod (input_0, input_1, ...). Element-wise multiply across all input short vectors **multiply _vff** : output = prod (input_0, input_1, ...). Element-wise multiply across all input float vectors.
Usage	**gr.multiply _vxx ()**
Parameters	
Note	

1.8.5.13) multiply_const_xx ( )

Type	Function
Description	**multiply_const_cc** : output = input * complex constant. Multiply constant by input complex streams. **multiply_const_ss** : output = input * short constant. Multiply constant by input short streams. **multiply_const_ii** : output = input * integer constant. Multiply constant by input integer streams. **multiply_const_ff** : output = input * float constant. Multiply constant by input float streams.
Usage	**gr.multiply_const _xx (k)**
Parameters	**k**: constant value, type depends on the function type
Note	
Sub Function 1	gr.multiply_const_xx.set_k(k): set the constant value on the fly.

1.8.5.14) multiply_const_vxx ( )

Type	Function
Description	**multiply_const_vcc** : output vector = input complex vector * constant complex vector (element-wise) **multiply_const_vii** : output vector = input integer vector * constant integer vector (element-wise) **multiply_const_vss** : output vector = input short vector * constant short vector (element-wise) **multiply_const_vff** : output vector = input float vector * constant float vector (element-wise)
Usage	**gr.multiply_const _vxx (k)**
Parameters	**k**: constant value, type depends on the function type
Note	
Sub Function 1	gr.multiply_const_vxx.set_k (k): set the constant value on the fly.
Sub Function 2	gr.multiply_const_vxx.k(): return the constant vector

1.8.5.15) mute_xx ( )

Type	Function
Description	**mute_cc**: output = input or zero if muted ,input is complex, output is complex **mute_ss**: output = input or zero if muted ,input is short, output is short **mute_ii**: output = input or zero if muted ,input is integer, output is integer **mute_ff**: output = input or zero if muted ,input is float, output is float
Usage	**gr.mute_xx (mute)**
Parameters	**mute** : bool, True or False
Note	
Sub Function 1	gr.mute_xx.set_mute(mute): set mute on the fly.
Sub Function 2	gr.mute_xx.mute(): return mute status, True or false

1.8.5.16) noise_source_x ( )

Type	Function
Description	**noise_source_c** : complex random number source with predefined distribution **noise_source_f** : float random number source with predefined distribution **noise_source_i** : integer random number source with predefined distribution **noise_source_s** : short random number source with predefined distribution
Usage	**gr.noise_source_x(type, ampl, seed)**
Parameters	**type** : GR_UNIFORM, GR_GAUSSIAN, GR_LAPLACIAN, GR_IMPULSE **ampl** : float, max signal amplitude **seed** : long, random function seed value
Note	Noise types should be used as follows : **gr.GR_UNIFORM** **gr.GR_GAUSSIAN** **gr.GR_LAPLACIAN** **gr.GR_IMPULSE**
Sub Function 1	gr.noise_source_x.set_type(type): set noise type.
Sub Function 2	gr.noise_source_x.set_amplitude(ampl): set float amplitude

1.8.5.17) peak_detector_xb ( )

Type	Function
Description	Detect the peak of a signal. If a peak is detected, this block outputs a 1, or it outputs 0's. **peak_detector_fb** : Float input stream. **peak_detector_ib** : Integer input stream.

	peak_detector_sb : Short input stream.
Usage	**gr.peak_detector_xb(threshold_factor_rise, threshold_factor_fall, look_ahead, alpha)**
Parameters	**threshold_factor_rise:** The threshold factor (float) determines when a peak has started. An average of the signal is calculated and when the value of the signal goes over threshold_factor_rise*average, we start looking for a peak. **threshold_factor_fall** :The threshold factor (float) determines when a peak has ended. An average of the signal is calculated and when the value of the signal goes bellow threshold_factor_fall*average, we stop looking for a peak. **look_ahead**: The look-ahead (integer) value is used when the threshold is found to look if there another peak within this step range. If there is a larger value, we set that as the peak and look ahead again. This is continued until the highest point is found with This look-ahead range. **alpha** : The gain value (float) of a moving average filter (Time Constant in sec)
Note	
Sub Function 1	gr.peak_detector_xb.set_threshold_factor_rise(thr): Set the threshold factor value for the rise time.
Sub Function 2	gr.peak_detector_xb.set_threshold_factor_fall(thr): Set the threshold factor value for the fall time.
Sub Function 3	gr.peak_detector_xb.set_look_ahead(look): Set the look ahead factor value
Sub Function 4	gr.peak_detector_xb.set_alpha(alpha): Set the running average alpha
Sub Function 5	gr.peak_detector_xb.threshold_factor_rise(): return the threshold factor value for the rise time.
Sub Function 6	gr.peak_detector_xb.threshold_factor_fall():return the threshold factor value for the fall time.
Sub Function 7	gr.peak_detector_xb._look_ahead():return the look ahead factor value
Sub Function 8	gr.peak_detector_xb.alpha():return the running average alpha

1.8.5.18) sample_and_hold_xx ( )

Type	Function
Description	Sample and hold circuit. Samples the data stream (input stream 0) and holds the value if the control signal is 1 (intput stream 1). **sample_and_hold_bb** : input stream is unsigned char **sample_and_hold_ff** : input stream is float **sample_and_hold_ii** : input stream is integer **sample_and_hold_ss** : input stream is short
Usage	**gr.sample_and_hold_xx()**
Parameters	
Note	

1.8.5.19) sig_source_x ( )

Type	Function
Description	**sig_source_c** : signal generator with gr_complex output. **sig_source_f** : signal generator with float output. **sig_source_i** : signal generator with integer output. **sig_source_s** : signal generator with short output.
Usage	**gr.sig_source_x(sampling_freq, waveform, frequency, ampl, offset)**
Parameters	**sampling_freq** : double **waveform** : *GR_CONST_WAVE GR_SIN_WAVE GR_COS_WAVE GR_SQR_WAVE GR_TRI_WAVE GR_SAW_WAVE* **frequency** : double, signal frequency **ampl** : double, signal max amplitude **offset** : DC offset, value type depends on signal type
Note	Waveform types should be used as follows : **gr.GR_CONST_WAVE** **gr.GR_SIN_WAVE** **gr.GR_COS_WAVE**

	gr.GR_SQR_WAVE **gr.GR_TRI_WAVE** **gr.GR_SAW_WAVE**
Sub Function 1	gr.sig_source_x.set_sampling_freq(sampling_freq): set sampling frequency
Sub Function 2	gr.sig_source_x.set_waveform(waveform): set signal waveform
Sub Function 3	gr.sig_source_x.set_frequency(frequency): set signal frequency
Sub Function 4	gr.sig_source_x.set_amplitude(ampl): set signal amplitude
Sub Function 5	gr.sig_source_x.set_offset(offset): set DC offset
Sub Function 6	gr.sig_source_x.sampling_freq(): return sampling frequency
Sub Function 7	gr.sig_source_x.set_waveform(waveform): return signal waveform
Sub Function 8	gr.sig_source_x.set_frequency(frequency): return signal frequency
Sub Function 9	gr.sig_source_x.set_amplitude(ampl): return signal amplitude
Sub Function 10	gr.sig_source_x.set_offset(offset): return DC offset

1.8.5.20) sub_xx ( )

Type	Function
Description	**sub _cc** : output = sub (input_0, input_1, ...). Subtruct across all input complex streams. **sub _ii** : output = sub (input_0, input_1, ...). Subtruct across all input integer streams. **sub _ss** : output = sub (input_0, input_1, ...). Subtruct across all input short streams. **sub _ff** : output = sub (input_0, input_1, ...). Subtruct across all input float streams.
Usage	**gr.sub _xx ()**
Parameters	
Note	

1.8.5.21) vector_sink_x ( )

Type	Function
Description	**vector_sink_f** : Float sink that writes to a vector. **vector_sink_c** : Complex sink that writes to a vector. **vector_sink_i** : Integer sink that writes to a vector. **vector_sink_s** : Short sink that writes to a vector. **vector_sink_b** : unsigned char sink that writes to a vector.
Usage	**gr.vector _sink_x ()**
Parameters	
Note	
Sub Function 1	gr.vector_sink_x .data() : Give us the stored vector data

1.8.5.22) vector_source_x ( )

Type	Function
Description	**vector_source_f** : Source of float that gets its data from a vector **vector_source_c** : Source of complex that get its data from a vector **vector_source_i** : Source of integer that gets its data from a vector **vector_source_s**: Source of short that gets its data from a vector **vector_source_b** : Source of unsigned char that gets its data from a vector
Usage	**gr.vector _source_x (data, repeat=false)**
Parameters	**data** : data to be used to form the vector, type depends on function type. **repeat**: bool True, or False, keep the source running by cyclicly repeating the data
Note	

1.8.6) gnuradio/ gr / gnuradio_swig_py_runtime.py

Type	Python file
Description	This file was automatically generated by SWIG. Some mathematical, source and sink blocks are defined here.
Examples	
Note	

1.8.6.1) io_signature ( )

Type	Function
Description	Create an i/o signature for input and output ports.
Usage	**gr.io_signature(min_streams, max_streams, sizeof_stream_item)**
Parameters	***min_streams*** : specify minimum number of streams (>= 0) ***max_streams*** : specify maximum number of streams (>= min_streams or -1 -> infinite) ***sizeof_stream_items*** : specify the size of the items in the streams
Note	
Sub Function 1	gr.io_signature.min_streams() : return min number of streams
Sub Function 2	gr.io_signature.max_streams() : return max number of streams
Sub Function 3	gr.io_signature.sizeof_stream_item() : return stream size

1.8.6.2) buffer ()

Type	Function
Description	Single writer, multiple reader fifo. Allocate a buffer that holds at least nitems of size sizeof_item. The total size of the buffer will be rounded up to a system dependent boundary. This is typically the system page size, but under MS windows is 64KB.
Usage	**gr.buffer(nitems, sizeof_item)**
Parameters	**nitem**: Integer, number of items **sizeof_item** : 8 if the data is complex, 4 if the data isinteger, 4 if the data is float, 2 if the data is short, 1 if the data is unsigned character.
Note	
Sub Function 1	gr.buffer.space_availaible () :Integer, return number of items worth of space available for writing
Sub Function 2	gr.buffer.write_pointer (): return pointer to write buffer.
Sub Function 3	gr.buffer.update_write_pointer():tell buffer that we wrote nitems into it
Sub Function 4	gr.buffer.set_done(true or false)
Sub Function 5	gr.buffer.done() : return True or false

1.8.6.3) buffer_reader ( )

Type	Function
Description	Used to let us keep track of the readers of a gr_buffer.
Usage	**gr.buffer_reader(buf, nzero_preload)**
Parameters	***buf*** *: gr_buffer pointer* ***nzero_preload*** : number of zero items to "preload" into buffer
Note	
Sub Function 1	gr.buffer_reader.items_availaible () :Integer, Return number of items available for reading
Sub Function 2	gr.buffer_reader.buffer ():Return buffer this reader reads from.
Sub Function 3	gr.buffer_reader.maximum_possible_items_available():Return maximum number of items that could ever be available for reading. This is used as a sanity check in the

	scheduler to avoid looping forever.
Sub Function 4	gr.buffer_reader.read_pointer() : return pointer to read buffer
Sub Function 5	gr.buffer_reader.update_read_pointer(nitems) : integer
Sub Function 6	gr.buffer_reader.set_done(true or false)
Sub Function 7	gr.buffer_reader.done() : return True or false

1.8.6.4) basic_block ( )

Type	Function
Description	The abstract base class for all signal processing blocks. Basic blocks are the bare abstraction of an entity that has a name and a set of inputs and outputs. These are never instantiated directly; rather, this is the abstract parent class of both gr_hier_block, which is a recursive container, and gr_block, which implements actual signal processing functions.
Usage	
Parameters	
Note	Needs more documentation
Sub Function 1	
Sub Function 2	

1.8.6.5) block ( )

Type	Function
Description	The abstract base class for all 'terminal' processing blocks. A signal processing flow is constructed by creating a tree of hierarchical blocks, which at any level may also contain terminal nodes that actually implement signal processing functions. This is the base class for all such leaf nodes. Blocks have a set of input streams and output streams. The input_signature and output_signature define the number of input streams and output streams respectively, and the type of the data items in each stream. Although blocks may consume data on each input stream at a different rate, all outputs streams must produce data at the same rate. That rate may be different from any of the input rates. User derived blocks override two methods, forecast and general_work, to implement their signal processing behavior. **forecast ()** is called by the system scheduler to determine how many items are required on each input stream in order to produce a given number of output items. **general_work ()** is called to perform the signal processing in the block. It reads the input items and writes the output items.
Usage	
Parameters	
Note	Needs more documentation
Sub Function 1	
Sub Function 2	

1.8.6.6) block_detail ( )

Type	Function
Description	Implementation details to support the signal processing abstraction. This class contains implementation detail that should be "out of sight" of almost all users of GNU Radio. This decoupling also means that we can make changes to the guts without having to recompile everything.
Usage	
Parameters	
Note	Needs more documentation
Sub Function 1	
Sub Function 2	

1.8.6.7) hier_block2 ( )

Type	Function
Description	New Hierarchical container class for gr_block's.
Usage	
Parameters	
Note	Needs more documentation
Sub Function 1	
Sub Function 2	

1.8.6.8) single_threaded_scheduler ( )

Type	Function
Description	Simple scheduler for stream computations.
Usage	
Parameters	
Note	Needs more documentation
Sub Function 1	
Sub Function 2	

1.8.6.9) message ( )

Type	Function
Description	Creat Message
Usage	**gr.message (type, arg1, arg2, length)**
Parameters	**type** :Long, message type usually =0 **arg1** : Double, any numeric argument **arg2** : Double, any numeric argument **length** : Message length in bytes
Note	Needs more documentation
Sub Function 1	gr.message.type() : return long
Sub Function 2	gr.message.arg1() : return double
Sub Function 3	gr.message.arg2() : return double
Sub Function 4	gr.message.set_type(type)
Sub Function 5	gr.message.set_arg1(arg1)
Sub Function 6	gr.message.set_arg2(arg2)
Sub Function 7	gr.message.length() : return message length
Sub Function 8	gr.message.msg (): Put the message here. Return the msg
Sub Function 9	gr.message.to_string() : Return the body of message as string

1.8.6.9) message_from_string ( )

Type	Function
Description	???????? Generate message from string
Usage	**gr.message_from_string(s, type, arg1, arg2)**
Parameters	**s** : String **type** :long **arg1** : Double, any numeric argument **arg2** : Double, any numeric argument
Note	Needs more documentation
Sub Function 1	gr.message from_string.type() : return long
Sub Function 2	gr.message from_string.arg1() : return double
Sub Function 3	gr.message from_string.arg2() : return double
Sub Function 4	gr.message from_string.set_type(type)

Sub Function 5	gr.message from_string.set_arg1(arg1)
Sub Function 6	gr.message from_string.set_arg2(arg2)
Sub Function 7	gr.message from_string.msg() : return pointer of type unsigned char to the message
Sub Function 8	gr.message from_string.to_string() : return string

1.8.6.10) message_handler ( )

Type	Function
Description	???????? Abstract class of message handlers
Usage	**gr.message_handler.handle(msg)**
Parameters	**msg** : handle
Note	Needs more documentation

1.8.6.11) msg_queue ( )

Type	Function
Description	Thread-safe message queue. Retuen a pointer to the created message queue
Usage	**gr.msg_queue(limit)**
Parameters	**limit** : Set the number of holded messages in the queue
Note	Needs more documentation
Sub Function 1	gr.msg_queue.handle (msg) : Generic msg_handler method: insert the message.
Sub Function 2	gr.msg_queue.insert_tail (msg) : Insert message at tail of queue.
Sub Function 3	gr.msg_queue.delete_head () : Delete message from head of queue and return it. Block if no message is available.
Sub Function 4	gr.msg_queue.delete_head_nowait () : If there's a message in the queue, delete it and return it. If no message is available, return 0.
Sub Function 5	gr.msg_queue.flush () : Delete all messages from the queue.
Sub Function 6	gr.msg_queue.empty_p () : is the queue empty?
Sub Function 7	gr.msg_queue.full_p () : is the queue full?
Sub Function 8	gr.msg_queue.count() : return (unsigned integer) number of messages in queue
Sub Function 9	gr.msg_queue.limit (): return (unsigned integer) limit on number of message in queue. 0 means unbounded

1.8.6.12) dispatcher ( )

Type	Function
Description	Invoke callbacks based on select.
Note	Needs more documentation
Sub Function 1	**gr.dispatcher.loop (timeout=10):** Event dispatching loop. Enter a polling loop that only terminates after all gr_select_handlers have been removed. timeout sets the timeout parameter to the select() call, measured in seconds. ***timeou****t:* maximum number of seconds to block in select.
Sub Function 2	gr.dispatcher.add_handler(handler) :
Sub Function 3	gr.dispatcher.del_handler(handler) :
Sub Function 4	gr.dispatcher.del_handler(handler) :

1.8.6.13) error_handler ( )

Type	Function
Description	Abstract for error handler
Note	Needs more documentation
Sub Function 1	

Sub Function 2	
Sub Function 3	
Sub Function 4	

1.8.6.14) file_error_handler ( )

Type	Function
Description	File error handler
Note	Needs more documentation

1.8.6.15) sync_block ( )

Type	Function
Description	Synchronous 1:1 input to output with history .Override work to provide the signal processing implementation.
Note	Needs more documentation

1.8.6.16) sync_decimator ( )

Type	Function
Description	Synchronous N:1 input to output with history. Override work to provide the signal processing implementation.
Note	Needs more documentation

1.8.6.16) sync_interpolator ( )

Type	Function
Description	Synchronous 1:N input to output with history. Override work to provide the signal processing implementation.
Note	Needs more documentation

1.8.6.17) top_block ( )

Type	Function
Description	Top-level hierarchical block representing a flowgraph.
Usage	**gr.top_block(name)**
Parameters	**name** : string
Note	Needs more documentation
Sub Function 1	gr.top_block.run () : The simple interface to running a flowgraph. Calls start () then wait () . Used to run a flowgraph that will stop on its own, or to run a flowgraph indefinitely until SIGINT is received.
Sub Function 2	gr.top_block.start () : Start the contained flowgraph. Creates one or more threads to execute the flow graph. Returns to the caller once the threads are created.
Sub Function 3	gr.top_block.stop () : Stop the running flowgraph. Notifies each thread created by the scheduler to shutdown, then returns to caller.
Sub Function 4	gr.top_block.wat () : Wait for a flowgraph to complete. Flowgraphs complete when either (1) all blocks indicate that they are done (typically only when using gr.file_source, or gr.head, or (2) after stop has been called to request shutdown.
Sub Function 5	gr.top_block.is_running() : Returns true if flowgraph is running

1.8.6.18) enable_realtime_scheduling ( )

Type	Function
Description	If possible, enable high-priority "real time" scheduling. **Return** gr.RT_Ok if successful,
Usage	**gr.enable_realtime_scheduling()**
Parameters	
Note	The possible Return values are : RT_NOT_IMPLEMENTED,RT_NO_PRIVS, RT_OTHER_ERROR

1.8.7) gnuradio/ gr / threading.py

Type	Python file
Description	Choose load gr_threading_23.py or gr_threading_24.py
Examples	
Note	

1.8.8) gnuradio/ gr / threading_23.py

Type	Python file
Description	Threading module for version 2.3
Examples	
Note	

1.8.9) gnuradio/ gr / threading_24.py

Type	Python file
Description	Threading module for version 2.4
Examples	
Note	

1.8.10) gnuradio/ gr / hier_block2.py

Type	Python file
Description	Construct new hierarchical blocks for flowgraph
Examples	
Note	

1.8.11) gnuradio/ gr / hier_block.py

Type	Python file
Description	Simple concrete class for building hierarchical blocks.This class assumes that there is at most a single block at the head of the chain and a single block at the end of the chain. Either head or tail may be None indicating a sink or source respectively. It can compose one or more blocks (primitive or hierarchical) into a new hierarchical block.
Usage	**gr.hier_block(fg, head_block, tail_block)**
Parameters	**fg**: The flow graph that contains this hierarchical block. type fg: flow_graph **head_block**: the first block in the signal processing chain. type head_block: None or subclass of gr.block or gr.hier_block_base

	tail_block: the last block in the signal processing chain. type tail_block: None or subclass of gr.block or gr.hier_block_base
Examples	
Note	

1.8.12) gnuradio/ gr / prefs.py

Type	Python file
Description	Base class for representing user preferences in the windows INI files. The real implementation is in Python, and is accessable from C++ via the magic of SWIG directors. Derive our 'real class' from the stubbed out base class that has support for SWIG directors. This allows C++ code to magically and transparently invoke the methods in this python class.
Sub Function 1	**gr.prefs ().has_section (section)** : Does section exist? Section is string, return bool True or False
Sub Function 2	**gr.prefs ().has_option (section, option)** : Does option exist? option is string, return bool True or False
Sub Function 3	**gr.prefs().get_string (section,option, default_val)** : If option exists return associated value; else return the string default_val.
Sub Function 4	**gr.prefs().get_bool (section,option, default_val)** : If option exists and value can be converted to bool, return it; else return the bool default_val.
Sub Function 5	**gr.prefs().get_long (section,option, default_val)** : If option exists and value can be converted to long, return it; else return the long default_val.
Sub Function 6	**gr.prefs().get_double (section,option, default_val)** : If option exists and value can be converted to double, return it; else return the double default_val.
Examples	See usrp_spectrum_sense.py
Note	Needs more documenation

1.8.13) gnuradio/ gr / scheduler.py

Type	Python file
Description	Schedule the threads.Invoke the single threaded scheduler's run method Note that we're in a new thread, and that sts_pyrun releases the global interpreter lock. This has the effect of evaluating the graph in parallel to the main line control code.
Examples	
Note	

1.8.14) gnuradio/ gr / top_block.py

Type	Python file
Description	This hack forces a 'has-a' relationship to look like an 'is-a' one. It allows Python classes to subclass this one, while passing through method calls to the C++ class shared pointer from SWIG.It also allows us to intercept method calls if needed. This allows the 'run_locked' methods, which are defined in gr_top_block.i, to release the Python global interpreter lock before calling the actual method in gr_top_block
Examples	
Note	Needs more documentation

1.8.15) gnuradio/ gr / gnuradio_swig_python.py

Type	Python file
Description	This file implements the old gnuradio_swig_python namespace
Examples	
Note	

1.8.16) gnuradio/ gr / gnuradio_swig_io.py

Type	Python file
Description	This file implements many sink and source blocks
Examples	
Note	

1.8.16.1) file_sink_base ( )

Type	Function
Description	Common base class for file sinks.
Usage	**gr.file_sink_base(filename, is_binary)**
Parameters	**filename**: File Name **is_binary** : bool True or False
Note	
Sub Function 1	gr.gr_file_sink_base.open (filename) : Open filename and begin output to it.
Sub Function 2	gr.gr_file_sink_base.close () : Close current output file. Closes current output file and ignores any output until open is called to connect to another file.
Sub Function 3	gr.gr_file_sink_base.do_update () : if we've had an update, do it now.

1.8.16.2) file_sink ( )

Type	Function
Description	Write a stream to a binary file.
Usage	**gr.file_sink(itemzize,filename)**
Parameters	**itemsize**: one of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char **filename**: File name
Note	

1.8.16.3) file_source ( )

Type	Function
Description	Read stream from binary file.
Usage	**gr.file_source(itemzize,filename, repeat)**
Parameters	**itemsize**: gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char filename: File name **repeat** : Bool True, or False, repeat file reading when EOF reached.
Note	
Sub Function 1	gr.file_source.seek (seek_point,whence) : seek file to seek_point relative to whence ***seek_point*** : sample offset in file *whence* : one of gr.SEEK_SET, gr.SEEK_CUR, gr.SEEK_END

1.8.16.4) file_descriptor_sink ( )

Type	Function
Description	Write stream to file descriptor.
Usage	**gr.file_descriptor_sink(itemzize, fd)**
Parameters	**itemsize**: one of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char **fd**: File descriptor, integer
Note	Needs more documenation

1.8.16.5) file_descriptor_source ( )

Type	Function
Description	Read stream from file descriptor.
Usage	**gr.file_descriptor_source(itemzize,fd, reapeat)**
Parameters	**itemsize**: gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char **fd**: File descriptor, integer **repeat** : Bool True, or False repeat file reading when EOF reached.
Note	Needs more documenation

1.8.16.6) microtune_xxxx_eval_board ( )

Type	Function
Description	Abstract class for controlling microtune xxxx eval board
Usage	
Parameters	
Note	Needs more documentation

1.8.16.7) microtune_4702_eval_board ( )

Type	Function
Description	Control microtune 4702 eval board
Usage	
Parameters	
Note	Needs more documentation

1.8.16.8) microtune_4937_eval_board ( )

Type	Function
Description	Control microtune 4937 eval board
Usage	
Parameters	
Note	Needs more documentation

1.8.16.8) sdr_1000_base ( )

Type	Function
Description	Very low level interface to SDR 1000 xcvr hardware. See sdr_1000.py for a higher level interface.
Usage	
Parameters	
Note	

1.8.16.9) oscope_sink_f ( )

Type	Function
Description	Building block for python oscilloscope module. Accepts 1 to 16 float streams.
Usage	**gr.oscope_sink_f(sampling_rate,msgq)**
Parameters	**sampling_rate** : Double represent sampling rate **msgq** : Message queue
Note	Needs more documentation

1.8.16.10) ppio ( )

Type	Function
Description	Abstract class that provides low level access to parallel port bits.
Usage	
Parameters	
Note	Needs more documentation

1.8.16.11) message_source ( )

Type	Function
Description	Turn received messages into a stream.
Usage	**gr.message_source(itemsize,msgq_limit)**
Parameters	**itemsize**: Size of data **msgq_limit** : Integer, number of messages to hold in the queue
Note	Needs more documentation

1.8.16.12) message_sink ( )

Type	Function
Description	Gather (convert) the received items into messages and insert into a message queue. Message type is 0, msg.arg1 will hold the itemsize, and msg.arg2 will hold number of items in the message.
Usage	**gr.message_sink(itemsize,msgq,dont_block)**
Parameters	**itemsize**: Size of data **msgq** : Message queue **don't_block** : bool True or False
Note	Needs more documentation

1.8.16.13) udp_sink ( )

Type	Function
Description	Write stream to an UDP socket.
Usage	**gr.udp_sink(itemsize, src, port_src, dst, port_dst, payload_size)**
Parameters	**itemsize** : The size (in bytes) of the item datatype **src** : The source address as either the host name or the 'numbers-and-dots' IP address **port_src** : Destination port to bind to (0 allows socket to choose an appropriate port) **dst** : The destination address as either the host name or the 'numbers-and-dots' IP address **port_dst** : Destination port to connect to **payload_size** : UDP payload size by default set to 1472 = (1500 MTU - (8 byte UDP header) - (20 byte IP header))
Note	
Sub Function 1	gr.udp_sink.open() : open a socket specified by the port and ip address info Opens a socket, binds to the address, and makes connectionless association over UDP. If any of these fail, the fuction retuns the error and exits.
Sub Function 2	gr.udp_sink.close () : Close current socket. Shuts down read/write on the socket
Sub Function 3	gr.udp_sink.payload_size() : return the PAYLOAD_SIZE of the socket

1.8.16.14) udp_source ( )

Type	Function
Description	Read stream from UDP socket.
Usage	**gr.udp_source(itemsize, src, port_src, payload_size)**
Parameters	**itemsize** : The size (in bytes) of the item datatype ***src*** : The source address as either the host name or the 'numbers-and-dots' IP address ***port_src*** : Destination port to bind to (0 allows socket to choose an appropriate port) ***payload_size*** : UDP payload size by default set to 1472 = (1500 MTU - (8 byte UDP header) - (20 byte IP header))
Note	
Sub Function 1	gr.udp_source.open() : open a socket specified by the port and ip address info Opens a socket, binds to the address, and makes connectionless association over UDP. If any of these fail, the fuction retuns the error and exits.
Sub Function 2	gr.udp_source.close () : Close current socket. Shuts down read/write on the socket
Sub Function 3	gr.udp_source.payload_size() : return the PAYLOAD_SIZE of the socket

1.8.17) gnuradio/ gr / gnuradio_swig_general.py

Type	Python file
Description	This file implements the general gnuradio blocks
Examples	
Note	

1.8.17.1) nop ( )

Type	Function
Description	Does nothing. Used for testing only.
Usage	**gr.nop(sizeof_stream_item)**

Parameters	**sizeof_stream_item**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char
Note	

1.8.27.2) null_sink ( )

Type	Function
Description	Null sink block.
Usage	**gr.null_sink (sizeof_stream_item)**
Parameters	**sizeof_stream_item**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char
Note	

1.8.27.3) null_source ( )

Type	Function
Description	Null source block. A source of zeros.
Usage	**gr.null_source (sizeof_stream_item)**
Parameters	**sizeof_stream_item**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char
Note	

1.8.27.4) head ( )

Type	Function
Description	Copies the first N items to the output then signals done.
Usage	**gr.head (sizeof_stream_item, nitems)**
Parameters	**sizeof_stream_item**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char **nitems** : Integer, number of samples to collect.
Note	

1.8.27.5) skiphead ( )

Type	Function
Description	Skips the first N items, from then on copies items to the output. Useful for building test cases and sources which have metadata or junk at the start
Usage	**gr.skiphead (sizeof_stream_item, nitems_to_skip)**
Parameters	**sizeof_stream_item: One of** gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char **nitems_to_skip** : Integer, number of samples to skip
Note	

1.8.27.6) quadrature_demod_cf ( )

Type	Function
Description	Quadrature demodulator: complex in, float out.This can be used to demod FM, FSK, GMSK, etc. The input is complex baseband.
Usage	**gr.quadrature_demod_cf(gain)**

Parameters	**gain** : float
Note	Needs more documenation

1.8.27.7) float_to_complex ( )

Type	Function
Description	Convert 1 or 2 streams of float to a stream of gr_complex.
Usage	**gr.float_to_complex()**
Parameters	
Note	

1.8.27.8) check_counting_s ( )

Type	Function
Description	Sink that checks if its input stream consists of a counting sequence. This sink is typically used to test the USRP "Counting Mode" or "Counting mode 32 bit".
Usage	**gr.check_counting_s(do_32bit)**
Parameters	***do_32bit*** : Bool True or False, expect an interleaved 32 bit counter in stead of 16 bit counter (default false)
Note	

1.8.27.9) lfsr_32k_source_s ( )

Type	Function
Description	LFSR pseudo-random source with period of 2^15 bits (2^11 shorts). This source is typically used along with gr_check_lfsr_32k_s to test the USRP using its digital loopback mode.
Usage	**gr.lfsr_32k_source_s()**
Parameters	
Note	

1.8.27.10) check_lfsr_32k_s ( )

Type	Function
Description	Sink that checks if its input stream consists of a lfsr_32k sequence. This sink is typically used along with gr_lfsr_32k_source_s to test the USRP using its digital loopback mode.
Usage	**gr.check_lfsr_32k_s()**
Parameters	
Note	
Sub Function 1	gr.check_lfsr_32k_s.ntotal() : Return long represent total number of elements
Sub Function 2	gr.check_lfsr_32k_s.nright() : Return long represent correct number of elements
Sub Function 3	gr.check_lfsr_32k_s.runlength () : Return long represent ????????????

1.8.27.11) stream_to_vector ( )

Type	Function
Description	Convert a stream of items into a stream of blocks containing nitems_per_block
Usage	**gr.stream_to_vector(item_size,nitems_per_block)**
Parameters	**item_size**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char **nitems_per_block** : vector length
Note	

1.8.27.12) vector_to_stream ( )

Type	Function
Description	Convert a stream of blocks of nitems_per_block items into a stream of items
Usage	**gr.vector_to_stream(item_size,nitems_per_block)**
Parameters	**item_size**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char **nitems_per_block** : vector length
Note	

1.8.27.13) keep_one_in_n ( )

Type	Function
Description	Decimate a stream, keeping one item of size (itemsize) out of every n.
Usage	**gr.keep_one_in_n(item_size, n)**
Parameters	**item_size**: Item size we wish to keep **n** : integer
Note	
Sub Function 1	gr.keep_one_in_n.set_n(n)

1.8.27.14) fft_vcc ( )

Type	Function
Description	Compute forward or reverse FFT, complex vector in / complex vector out.
Usage	**gr.fft_vcc(fft_size,forward, window, shift=false)**
Parameters	**fft_size** : integer **forward** : bool True for forward FFT, False for inverse FFT **window** : window vector, **shift** : bool True or false
Note	
Sub Function 1	gr.fft_vcc.set_window(window)
Example	See usrp_spectrum_sense.py

1.8.27.15) fft_vfc ( )

Type	Function
Description	Compute forward or reverse FFT, float vector in / complex vector out.
Usage	**gr.fft_vfc(fft_size,forward, window)**
Parameters	**fft_size** : integer

	forward : bool True for forward FFT, False for inverse FFT **window** : window vector
Note	
Sub Function 1	gr.fft_vfc.set_window(window)

1.8.27.16) float_to_short ( )

Type	Function
Description	Convert stream of float to a stream of short.
Usage	**gr.float_to_short()**
Parameters	
Note	

1.8.27.17) float_to_uchar ( )

Type	Function
Description	Convert stream of float to a stream of unsigned character.
Usage	**gr.float_to_uchar()**
Parameters	
Note	

1.8.27.18) short_ to_float ( )

Type	Function
Description	Convert stream of short to a stream of float.
Usage	**gr.short_to_float()**
Parameters	
Note	

1.8.27.19) char_to_float ( )

Type	Function
Description	Convert stream of characters to a stream of float.
Usage	**gr.char_to_float()**
Parameters	
Note	

1.8.27.20) uchar_to_float ( )

Type	Function
Description	Convert stream of unsigned characters to a stream of float.
Usage	**gr.uchar_to_float()**
Parameters	
Note	

1.8.27.21) frequency_modulator_fc ( )

Type	Function
Description	Frequency modulator block. float input; complex baseband output
Usage	**gr.frequency_modulator_fc(sensitivity)**
Parameters	**sensitivity** : double
Note	

1.8.27.22) phase_modulator_fc ( )

Type	Function
Description	Phase modulator block. output=complex(cos(in*sensitivity),sin(in*sensitivity))
Usage	**gr.phase_modulator_fc(sensitivity)**
Parameters	**sensitivity** : double
Note	

1.8.27.23) bytes_to_syms ( )

Type	Function
Description	Convert stream of bytes to stream of +/- 1 symbols (Turn it to NRZ data format). Input is a stream of bytes; output: stream of float. The combination of gr_packed_to_unpacked_bb followed by gr_chunks_to_symbols_bf or gr_chunks_to_symbols_bc handles the general case of mapping from a stream of bytes into arbitrary float or complex symbols.
Usage	**gr.bytes_to_syms()**
Parameters	
Note	

1.8.27.24) simple_framer ( )

Type	Function
Description	add sync field, seq number and command field to payload
Usage	**gr.simple_framer(payload_bytesize)**
Parameters	**payload_bytesize** : Integer
Note	Needs more documenation

1.8.27.25) simple_correlator ( )

Type	Function
Description	Inverse of gr_simple_framer (more or less).
Usage	**gr.simple_framer(payload_bytesize)**
Parameters	**payload_bytesize** : Integer
Note	Needs More documenation

1.8.27.26) align_on_samplenumbers_ss ( )

Type	Function
Description	Align several complex short (interleaved short) input channels with corresponding unsigned 32 bit sample_counters (provided as interleaved 16 bit values). Pay attention on how you connect this block. It expects a minimum of 2 usrp_source_s with nchan number of channels and FPGA_MODE_COUNTING_32BIT enabled. This means that the first complex_short channel on every input is an interleaved 32 bit counter. The samples are aligned by dropping samples untill the samplenumbers match.
Usage	**gr.align_on_samplenumbers_ss(nchan, align_interval)**
Parameters	**nchan** : of complex_short input channels (including the 32 bit counting channel) ***align_interval*** : is after how much samples (minimally) the sample-alignement is refreshed. Default is 128. A bigger value means less processing power but also requests more buffer space, which has a maximum. Decrease the align_interval if you get an error like: "sched: <gr_block align_on_samplenumbers_ss (0)> is requesting more input data than we can provide. ninput_items_required = 32768 max_possible_items_available = 16383 If this is a filter, consider reducing the number of taps."
Note	Needs More documenation

1.8.27.27) complex_to_float ( )

Type	Function
Description	Convert a stream of gr_complex to 1 or 2 streams of float
Usage	**gr.complex_to_float(vlen)**
Parameters	***vlen*** :vector len (default 1)
Note	

1.8.27.28) complex_to_real ( )

Type	Function
Description	Complex in, real part out (float)
Usage	**gr.complex_to_real(vlen)**
Parameters	***vlen*** :vector len (default 1)
Note	

1.8.27.29) complex_to_imag ( )

Type	Function
Description	Complex in, imaginary part out (float)
Usage	**gr.complex_to_imag(vlen)**
Parameters	***vlen*** :vector len (default 1)
Note	

1.8.27.30) complex_to_mag ( )

Type	Function
Description	Complex in, magnitude out (float)
Usage	**gr.complex_to_mag(vlen)**
Parameters	***vlen*** :vector len (default 1)
Note	

1.8.27.31) complex_to_mag_squared ( )

Type	Function
Description	Complex in, magnitude squared out (float)
Usage	**gr.complex_to_mag_squared(vlen)**
Parameters	***vlen*** :vector len (default 1)
Note	

1.8.27.32) complex_to_arg ( )

Type	Function
Description	complex in, angle out (float)
Usage	**gr.complex_to_arg(vlen)**
Parameters	***vlen*** :vector len (default 1)
Note	

1.8.27.33) complex_to_interleaved_short ( )

Type	Function
Description	Convert stream of complex to a stream of interleaved shorts.
Usage	**gr.complex_to_interleaved_short()**
Parameters	
Note	

1.8.27.34) interleaved_short_to_complex ( )

Type	Function
Description	Convert stream of interleaved shorts to a stream of complex.
Usage	**gr.interleaved_short_to_ complex ()**
Parameters	
Note	

1.8.27.35) firdes ( )

Type	Function
Description	Finite Impulse Response (FIR) filter design functions.
Note	

1.8.27.35.1) firdes.low_pass ( )

Type	Sub Function
Description	Design low pass FIR filter by using "window method"
Usage	**gr.firdes. low_pass (gain,sampling_freq,cutoff_freq, transition_width, window = WIN_HAMMING,beta = 6.76)**
Parameters	**gain**: overall gain of filter (typically 1.0) **sampling_freq**: sampling freq (Hz) **cutoff_freq**: center of transition band (Hz)

	transition_width: width of transition band (Hz). The normalized width of the transition band is what sets the number of taps required. Narrow --> more taps **window**: What kind of window to use. Determines maximum attenuation and passband ripple. Available window types are:WIN_HAMMING, WIN_HANN , WIN_BLACKMAN ,WIN_RECTANGULAR ,WIN_KAISER **beta**: parameter for Kaiser window (used only for Kaiser)
Note	**See firdes.window** for windowing information

1.8.27.35.2) firdes.high_pass ( )

Type	Sub Function
Description	Design high pass FIR filter by using "window method"
Usage	**gr.firdes. high_pass (gain,sampling_freq,cutoff_freq, transition_width, window = WIN_HAMMING,beta = 6.76)**
Parameters	**gain**: overall gain of filter (typically 1.0) **sampling_freq**: sampling freq (Hz) **cutoff_freq**: center of transition band (Hz) **transition_width**: width of transition band (Hz). The normalized width of the transition band is what sets the number of taps required. Narrow --> more taps **window**: What kind of window to use. Determines maximum attenuation and passband ripple. Available window types are:WIN_HAMMING, WIN_HANN , WIN_BLACKMAN ,WIN_RECTANGULAR ,WIN_KAISER **beta**: parameter for Kaiser window (used only for Kaiser)
Note	**See firdes.window** for windowing information

1.8.27.35.3) firdes.band_pass ( )

Type	Sub Function
Description	Design band pass FIR filter by using "window method"
Usage	**gr.firdes.band_pass (gain,sampling_freq,low_cutoff_freq, high_cutoff_freq, transition_width, window = WIN_HAMMING,beta = 6.76)**
Parameters	**gain**: overall gain of filter (typically 1.0) **sampling_freq**: sampling freq (Hz) **low_cutoff_freq**: center of low transition band (Hz) **high_cutoff_freq**: center of high transition band (Hz) **transition_width**: width of transition band (Hz). The normalized width of the transition band is what sets the number of taps required. Narrow --> more taps **window**: What kind of window to use. Determines maximum attenuation and passband ripple. Available window types are:WIN_HAMMING, WIN_HANN , WIN_BLACKMAN ,WIN_RECTANGULAR ,WIN_KAISER **beta**: parameter for Kaiser window (used only for Kaiser)
Note	**See firdes.window for windowing information**

1.8.27.35.4) firdes.complex_band_pass ( )

Type	Sub Function
Description	Design complex band pass FIR filter by using "window method"
Usage	**gr.firdes.complex_band_pass (gain,sampling_freq,low_cutoff_freq, high_cutoff_freq, transition_width, window = WIN_HAMMING,beta = 6.76)**
Parameters	**gain**: overall gain of filter (typically 1.0) **sampling_freq**: sampling freq (Hz)

	low_cutoff_freq: center of low transition band (Hz) **high_cutoff_freq**: center of high transition band (Hz) **transition_width**: width of transition band (Hz). The normalized width of the transition band is what sets the number of taps required. Narrow --> more taps **window**: What kind of window to use. Determines maximum attenuation and passband ripple. Available window types are:WIN_HAMMING, WIN_HANN , WIN_BLACKMAN ,WIN_RECTANGULAR ,WIN_KAISER **beta**: parameter for Kaiser window (used only for Kaiser)
Note	**See firdes.window for windowing information**

1.8.27.35.5) firdes.band_reject ( )

Type	Sub Function
Description	Design band reject FIR filter by using "window method"
Usage	**gr.firdes.band_reject (gain,sampling_freq,low_cutoff_freq, high_cutoff_freq, transition_width, window = WIN_HAMMING,beta = 6.76)**
Parameters	**gain**: overall gain of filter (typically 1.0) **sampling_freq**: sampling freq (Hz) **low_cutoff_freq**: center of low transition band (Hz) **high_cutoff_freq**: center of high transition band (Hz) **transition_width**: width of transition band (Hz). The normalized width of the transition band is what sets the number of taps required. Narrow --> more taps **window**: What kind of window to use. Determines maximum attenuation and passband ripple. Available window types are:WIN_HAMMING, WIN_HANN , WIN_BLACKMAN ,WIN_RECTANGULAR ,WIN_KAISER **beta**: parameter for Kaiser window (used only for Kaiser)
Note	See firdes.window for windowing information

1.8.27.35.6) firdes.hilbert ( )

Type	Sub Function
Description	Design Hilbert Transform FIR filter by using "window method"
Usage	**gr.firdes.hilbert (ntaps=19, windowtype = WIN_ RECTANGULAR,beta = 6.76)**
Parameters	**ntaps:** Number of taps, must be odd **windowtype**: What kind of window to use. Determines maximum attenuation and passband ripple. Available window types are:WIN_HAMMING, WIN_HANN , WIN_BLACKMAN ,WIN_RECTANGULAR ,WIN_KAISER **beta**: parameter for Kaiser window (used only for Kaiser)
Note	**See firdes.window for windowing information**

1.8.27.35.7) firdes.root_raised_cosine ( )

Type	Sub Function
Description	Design a root raised cosine FIR filter
Usage	**gr.firdes.root_raised_cosine (gain, sampling_freq, symbol_rate, alpha, taps)**
Parameters	**gain**: overall gain of filter (typically 1.0) **sampling_freq**: sampling freq (Hz) **symbol_rate**: symbol rate NOT bitrate (unless BPSK), must be a factor of sample rate **alpha**: excess bandwidth factor **ntaps**: number of taps
Note	

1.8.27.35.8) firdes.gaussian ( )

Type	Sub Function
Description	Design a gaussian FIR filter
Usage	**gr.firdes.gaussian (gain, spb, bt, ntaps)**
Parameters	**gain**: overall gain of filter (typically 1.0) **spb**: symbols per bit, symbol rate, must be a factor of sample rate **bt** : Bandwidth to bit rate ratio (bandwidth * symbol time) **ntaps**: number of taps
Note	

1.8.27.35.9) firdes.window ( )

Type	Sub Function
Description	Window taps maker
Usage	**gr.firdes.window (type, ntaps, beta)**
Parameters	**type**: window type, one of : WIN_HAMMING : Maximum Attenuation **53dB** WIN_HANN : Maximum Attenuation **44dB** WIN_BLACKMAN : Maximum Attenuation **74dB** WIN_RECTANGULAR , WIN_KAISER : max attenuation a function of beta, google it **ntaps**: number of taps **beta**: parameter for Kaiser window (used only for Kaiser)
Note	The usage of these window types is as followes : **gr.firdes. WIN_HAMMING** **gr.firdes. WIN_HANN** **gr.firdes. WIN_BLACKMAN** **gr.firdes. WIN_RECTANGULAR ,** **gr.firdes. WIN_KAISER**

1.8.27.36) interleave ()

Type	Function
Description	Interleave N inputs to a single output
Usage	**gr.interleave(item_size)**
Parameters	**Item_size**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char
Note	

1.8.27.37) deinterleave ()

Type	Function
Description	Deinterleave a single input into N outputs
Usage	**gr.deinterleave(item_size)**
Parameters	**Item_size**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char
Note	

1.8.27.38) delay ()

Type	Function
Description	Delay the input by a certain number of samples
Usage	**gr.delay(itemsize, delay)**
Parameters	**Itemsize**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char **delay** : Integer, number of samples
Note	
Sub Function 1	gr.delay.set_delay (delay) : Set block delay.
Sub Function 2	gr.delay.delay () : Return block delay.

1.8.27.39) simple_sequelch_cc ()

Type	Function
Description	Simple squelch block based on average signal power and threshold in dB. Output equal input if not muted.
Usage	**gr.simple_squelch_cc(threshold_db, alpha)**
Parameters	**threshold_db** : double **alpha** : Double , the gain value of a moving average filter (Time Constant in sec)
Note	Needs more documenation
Sub Function 1	gr.simple_sequelch_cc.set_threshold(decibels)
Sub Function 2	gr.simple_sequelch_cc.set_alpha(alpha)
Sub Function 3	gr.simple_sequelch_cc.threshold() : Return block threshold
Sub Function 4	gr.simple_sequelch_cc.unmuted() : Return bool True or False
Sub Function 5	gr.simple_sequelch_cc.squelch_range() : Return float vector represents sequelch range

1.8.27.40) agc_xx ()

Type	Function
Description	High performance Automatic Gain Control class. **agc_cc** : The Power is calculated by the absolute value of the complex number. **agc_ff** : Power is approximated by absolute value.
Usage	**gr.agc_xx (rate, refrence, gain, max_gain)**
Parameters	**rate** :float (Time Constant in Sec) **refrence** : float refrence power **gain** : float Initial gain **max_gain** : float maximum gain
Note	Needs more documentation

1.8.27.41) gri_agc_xx ()

Type	Function
Description	High performance Automatic Gain Control class **gri_agc_cc**: The Power is calculated by the absolute value of the complex number. **gri_agc_ff** : Power is approximated by absolute value
Usage	**gr.gri_agc_xx(rate=1e-4, refrence=1.0, gain=1.0, max_gain=0.0)**
Parameters	**rate** :float (Time Constant in Sec) **refrence** : float refrence power **gain** : float Initial gain **max_gain** : float maximum gain

Note	Needs more documentation
Sub Function 1	gr.gri_agc_xx.rate() : Return rate
Sub Function 2	gr.gri_agc_xx.refrence() : Return refrence
Sub Function 3	gr.gri_agc_xx.gain() : Return gain
Sub Function 4	gr.gri_agc_xx.max_gain() : Return max gain
Sub Function 5	gr.gri_agc_xx.set_rate() : Set rate
Sub Function 6	gr.gri_agc_xx.set_refrence() : Set refrence
Sub Function 7	gr.gri_agc_xx.set_gain() : Set gain
Sub Function 8	gr.gri_agc_xx.set_max_gain() : Set max gain
Sub Function 9	gr.gri_agc_xx.scale (input) : ?????????????
Sub Function 10	gr.gri_agc_xx.scaleN (output [ ], input [ ], n) : ?????????????

1.8.27.42) gri_agc2_xx ()

Type	Function
Description	High performance Automatic Gain Control class **gri_agc2_cc** : For Power the absolute value of the complex number is used. **gri_agc2_ff** : Power is approximated by absolute value
Usage	**gr.gri_agc2_xx(attack_rate=1e-1, decay_rate=1e-2,refrence=1, gain=1, max_gain=0.0)**
Parameters	**attack_rate** :float **decay_rate** : float **refrence** : float refrence power **gain** : float initial gain **max_gain** : float
Note	Needs more documentation
Sub Function 1	gr.gri_agc_xx.attack_rate() : Return attack_rate
Sub Function 2	gr.gri_agc_xx.refrence() : Return refrence
Sub Function 3	gr.gri_agc_xx.gain() : Return gain
Sub Function 4	gr.gri_agc_xx.max_gain() : Return max gain
Sub Function 5	gr.gri_agc_xx.set_attack_rate() : Set attack_rate
Sub Function 6	gr.gri_agc_xx.set_refrence() : Set refrence
Sub Function 7	gr.gri_agc_xx.set_gain() : Set gain
Sub Function 8	gr.gri_agc_xx.set_max_gain() : Set max gain
Sub Function 9	gr.gri_agc_xx.scale (input) : ?????????????
Sub Function 10	gr.gri_agc_xx.scaleN (output [ ], input [ ], n) : ?????????????
Sub Function 11	gr.gri_agc_xx.decay_rate() : Return decay_rate
Sub Function 12	gr.gri_agc_xx.set_decay_rate() : Set decay_rate

1.8.27.43) rms_xx ()

Type	Function
Description	RMS average power. **rms_cf** : Input is complex, output is float **rms_ff** : Input is float, output is float.
Usage	**gr.rms_xx(alpha)**
Parameters	**alpha** : Double , the gain value of a moving average filter (Time Constant in sec)
Note	Needs more documentation
Sub Function 1	gr.rms_xx.unmuted() : Return bool True or False
Sub Function 2	gr.rms_xx_set_alpha(alpha) : Set alpha

1.8.27.44) nlog10_ff ()

Type	Function
Description	Output = n*log10(input) + k
Usage	**gr.nlog10_ff(n, vlen, k)**
Parameters	**n** : Float **vlen** : Unsigned Integer vector length **k** : float
Note	

1.8.27.45) fake_channel_encoder_pp ()

Type	Function
Description	Pad packet with alternating 1,0 pattern. Input: stream of byte vectors; output: stream of byte vectors
Usage	**gr.fake_channel_encoder_pp(input_vlen, output_vlen)**
Parameters	**input_vlen** : Integer **output_vlen** : Integer
Note	

1.8.27.46) fake_channel_decoder_pp ()

Type	Function
Description	Remove fake padding from packet. Input: stream of byte vectors; output: stream of byte vectors
Usage	**gr.fake_channel_decoder_pp(input_vlen, output_vlen)**
Parameters	**input_vlen** : Integer **output_vlen** : Integer
Note	

1.8.27.47) throttle ()

Type	Function
Description	Throttle flow of samples such that the average rate does not exceed samples_per_sec. Input: one stream of itemsize; output: one stream of itemsize
Usage	**gr.throttle(item_size, samples_per_sec)**
Parameters	**itemsize**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char **samples_per_sec** : Double
Note	

1.8.27.48) mpsk_receiver_cc ()

Type	Function
Description	This block takes care of receiving M-PSK modulated signals through phase, frequency,

	and symbol synchronization. It performs carrier frequency and phase locking as well as symbol timing recovery. It works with (D) BPSK, (D)QPSK, and (D)8PSK as tested currently. It should also work for OQPSK and PI/4 DQPSK. The phase and frequency synchronization are based on a Costas loop that finds the error of the incoming signal point compared to its nearest constellation point. The frequency and phase of the NCO are updated according to this error. There are optimized phase error detectors for BPSK and QPSK, but 8PSK is done using a brute-force computation of the constellation points to find the minimum. The symbol synchronization is done using a modified Mueller and Muller circuit from the paper: G. R. Danesfahani, T.G. Jeans, "Optimisation of modified Mueller and Muller algorithm," Electronics Letters, Vol. 31, no. 13, 22 June 1995, pp. 1032 - 1033. This circuit interpolates the downconverted sample (using the NCO developed by the Costas loop) every mu samples, then it finds the sampling error based on this and the past symbols and the decision made on the samples. Like the phase error detector, there are optimized decision algorithms for BPSK and QPKS, but 8PSK uses another brute force computation against all possible symbols. The modifications to the M&M used here reduce self-noise.
Usage	**gr.mpsk_receiver_cc(M, theta, alpha, beta, fmin, fmax, mu, gain_mu, omega, gain_omega, omega_rel)**
Parameters	***M*** : Modulation order of the M-PSK modulation. The constructor also chooses which phase detector and decision maker to use in the work loop based on the value of M. ***theta*** : Any constant phase rotation from the real axis of the constellation *alpha* gain parameter to adjust the phase in the Costas loop (~0.01) ***beta*** : Gain parameter to adjust the frequency in the Costas loop (~alpha^2/4) ***fmin*** : Minimum normalized frequency value the loop can achieve ***fma****x* : Maximum normalized frequency value the loop can achieve ***mu*** : Initial parameter for the interpolator [0,1] ***gain_mu*** : Gain parameter of the M&M error signal to adjust mu (~0.05) ***omega*** :Initial value for the number of symbols between samples (~number of samples/symbol) ***gain_omega*** : Gain parameter to adjust omega based on the error (~omega^2/4) ***omega_rel*** :Sets the maximum (omega*(1+omega_rel)) and minimum (omega*(1+omega_rel)) omega (~0.005)
Note	
Sub Function 1	gr.mpsk_receiver_cc.mu() : (M&M) Returns current value of mu
Sub Function 2	gr.mpsk_receiver_cc.omega() : (M&M) Returns current value of omega
Sub Function 3	gr.mpsk_receiver_cc.gain_mu() : M&M) Returns mu gain factor
Sub Function 4	gr.mpsk_receiver_cc.gain_omega() : (M&M) Returns omega gain factor
Sub Function 5	gr.mpsk_receiver_cc.set_mu() : (M&M) Set value of mu
Sub Function 6	gr.mpsk_receiver_cc. set_omega() : (M&M) Set value of omega
Sub Function 7	gr.mpsk_receiver_cc. set_gain_mu() : M&M) Set mu gain factor
Sub Function 8	gr.mpsk_receiver_cc. set_gain_omega() : (M&M) Set omega gain factor
Sub Function 9	gr.mpsk_receiver_cc.alpha() : (CL) Returns the value for alpha (the phase gain term)
Sub Function 10	gr.mpsk_receiver_cc.beta() : (CL) Returns the value of beta (the frequency gain term)
Sub Function 11	gr.mpsk_receiver_cc.freq() : (CL) Returns the current value of the frequency of the NCO in the Costas loop
Sub Function 12	gr.mpsk_receiver_cc.phase() : (CL) Returns the current value of the phase of the NCO in the Costal loop
Sub Function 13	gr.mpsk_receiver_cc.set_alpha() : (CL) Sets the value for alpha (the phase gain term)
Sub Function 14	gr.mpsk_receiver_cc. set_beta() : (CL) (CL) Setss the value of beta (the frequency gain term)
Sub Function 15	gr.mpsk_receiver_cc. set_freq() : (CL) (CL) Sets the current value of the frequency of the NCO in the Costas loop
Sub Function 16	gr.mpsk_receiver_cc. set_phase() : (CL) Setss the current value of the phase of the NCO in the Costal loop

1.8.27.49) stream_mux ()

Type	Function
Description	Creates a stream muxing block to multiplex many streams into one with a specified format. Muxes N streams together producing an output stream that contains N0 items from the first stream, N1 items from the second, etc. and repeats:[N0, N1, N2, ..., Nm, N0, N1, ...]
Usage	**gr.stream_mux(item_size, lengths)**
Parameters	***itemsize*** : The item size of the stream ***length****s* : A vector (list/tuple) specifying the number of items from each stream the mux together. Warning: this requires that at least as many items per stream are available or the system will wait indefinitely for the items.
Note	

1.8.27.50) stream_to_streams ()

Type	Function
Description	Convert a stream of items into a N streams of items. Converts a stream of N items into N streams of 1 item. Repeat and infinitum
Usage	**gr.stream_to_streams(item_size, nstreams)**
Parameters	***itemsize*** : The item size of the stream **nstreams** : Number of streams
Note	

1.8.27.51) streams_to_stream ()

Type	Function
Description	Convert N streams of 1 item into a 1 stream of N items. Convert N streams of 1 item into 1 stream of N items. Repeat and infinitum.
Usage	**gr.streams_to_stream(item_size, nstreams)**
Parameters	***itemsize*** : The item size of the stream **nstreams** : Number of streams
Note	

1.8.27.52) streams_to_vector ()

Type	Function
Description	Convert N streams of items to 1 stream of vector length N
Usage	**gr.streams_to_vector(item_size, nstreams)**
Parameters	***itemsize*** : The item size of the stream **nstreams** : Number of streams
Note	

1.8.27.53) stream_to_vector ()

Type	Function
Description	Convert a stream of items into a stream of blocks containing nitems_per_block
Usage	**gr.stream_to_vector(item_size, nitems_per_block)**
Parameters	***itemsize*** : The item size of the stream

	nitems_per_block : Number of items in the vector
Note	

1.8.27.54) vector_to_ streams ()

Type	Function
Description	Convert 1 stream of vectors of length N to N streams of items.
Usage	**gr.vector_to_streams(item_size, nstreams)**
Parameters	***itemsize*** : The item size of the stream **nstreams** : Number of streams
Note	

1.8.27.55) vector_to_stream ()

Type	Function
Description	Convert a stream of blocks of nitems_per_block items into a stream of items
Usage	**gr. vector_to_stream(item_size, nitems_per_block)**
Parameters	***itemsize*** : The item size of the stream **nitems_per_block** : Number of items in the vector
Note	

1.8.27.56) conjugate_cc ()

Type	Function
Description	output = complex conjugate of input
Usage	**gr. conjugate_cc()**
Parameters	
Note	

1.8.27.57) vco_f ()

Type	Function
Description	VCO - Voltage controlled oscillator. input: float stream of control voltages; output: float oscillator output
Usage	**gr. vco_f(sampling_rate, sensitivity, amplitude)**
Parameters	***sampling_rate*** : sampling rate (Hz) ***sensitivity*** : units are radians/sec/volt ***amplitude*** : output amplitude
Note	

1.8.27.58) threshold_ff ()

Type	Function
Description	??????????????????????

Usage	**gr. threshold_ff(lo,hi,initial_state)**
Parameters	**lo** : Low threshold value **hi** : High threshold value **initial_state**: ???????????????
Note	Needs more documentation
Sub Function 1	gr.threshold_ff.lo() :
Sub Function 2	gr.threshold_ff.hi() :
Sub Function 3	gr.threshold_ff.last_state() :
Sub Function 4	gr.threshold_ff.set_lo() :
Sub Function 5	gr.threshold_ff.set_hi() :
Sub Function 6	gr.threshold_ff.set_last_state() :

1.8.27.59) clock_recovery_mm_xx ()

Type	Function
Description	This implements the Mueller and Müller (M&M) discrete-time error-tracking synchronizer. The clock recovery block trucks the symbol clock and resamples as needed. The output of the block is a stream of soft symbols. The complex version here is based on: Modified Mueller and Muller clock recovery circuit Based: G. R. Danesfahani, T.G. Jeans, "Optimisation of modified Mueller and Muller algorithm," Electronics Letters, Vol. 31, no. 13, 22 June 1995, pp. 1032 - 1033. **clock_recovery_mm_cc :** Mueller and Müller (M&M) based clock recovery block with complex input, complex output. **clock_recovery_mm_ff :** Mueller and Müller (M&M) based clock recovery block with float input, float output.
Usage	**gr. clock_recovery_mm_xx(omega, gain_omega, mu, gain_mu, omega_relative_limit)**
Parameters	***omega*** :initial value for the number of symbols between samples (~number of samples/symbol) ***gain_omega*** : Gain parameter to adjust omega based on the error ***mu*** : Initial parameter for the interpolator ***gain_mu*** : Gain parameter of the M&M error signal to adjust mu ***omega_relative_limit*** :Sets the maximum and minimum omega
Note	Needs more documentation
Sub Function 1	gr. clock_recovery_mm_xx.omega() : Return omega
Sub Function 2	gr. clock_recovery_mm_xx.mu() : Return mu
Sub Function 3	gr. clock_recovery_mm_xx.gain_omega() : Return gain_omega
Sub Function 4	gr. clock_recovery_mm_xx.gain_mu() : Return gain_mu
Sub Function 5	gr. clock_recovery_mm_xx.set_omega(omega) : Set omega
Sub Function 6	gr. clock_recovery_mm_xx.set_mu(mu) : Set mu
Sub Function 7	gr. clock_recovery_mm_xx.set_gain_omega(gain_omega) : Set gain_omega
Sub Function 8	gr. clock_recovery_mm_xx.set_gain_mu(gain_mu) : Set gain_mu
Sub Function 9	gr. clock_recovery_mm_xx.set_verbose(verbose) : Set printing

1.8.27.60) dd_mpsk_sync_cc ()

Type	Function
Description	Decision directed M-PSK synchronous demod This block performs joint carrier tracking and symbol timing recovery. Input: complex baseband; output: properly timed complex samples ready for slicing. At this point, it handles only QPSK.
Usage	**gr.dd_mpsk_sync_cc(alpha, beta, max_freq, min_freq, ref_phase, omega, gain_omega,mu, gain_mu)**
Parameters	***alpha*** : Gain parameter to adjust the phase in the Costas loop ***beta*** : Gain parameter to adjust the frequency in the Costas loop

	min_freq : Minimum normalized frequency value the loop can achieve ***max_freq*** : Maximum normalized frequency value the loop can achieve **ref_phase**: ?????????????? ***mu*** : Initial parameter for the interpolator ***gain_mu*** : Gain parameter of the M&M error signal to adjust mu ***omega*** :Initial value for the number of symbols between samples (~number of samples/symbol) ***gain_omega*** : Gain parameter to adjust omega based on the error
Note	Needs more documentation
Sub Function 1	gr.mpsk_ sync _cc.mu() : (M&M) Returns current value of mu
Sub Function 2	gr.mpsk_ sync_cc.omega() : (M&M) Returns current value of omega
Sub Function 3	gr.mpsk_ sync _cc.gain_mu() : M&M) Returns mu gain factor
Sub Function 4	gr.mpsk_ sync _cc.gain_omega() : (M&M) Returns omega gain factor
Sub Function 5	gr.mpsk_ sync _cc.set_mu() : (M&M) Set value of mu
Sub Function 6	gr.mpsk_ sync _cc. set_omega() : (M&M) Set value of omega
Sub Function 7	gr.mpsk_ sync _cc. set_gain_mu() : M&M) Set mu gain factor
Sub Function 8	gr.mpsk_ sync _cc. set_gain_omega() : (M&M) Set omega gain factor

1.8.27.61) packet_sink ()

Type	Function
Description	Process received bits looking for packet sync, header, and process bits into packet
Usage	**gr.packet_sink(sync_vector, target_queue, threshold)**
Parameters	**sync_vector**: vector of unsigned charaters. **target_gueue** : message queue **threshold** : Integer
Note	Needs more documentation
Sub Function 1	gr.packet_sink.carrier_sensed() : Return true if we detect carrier

1.8.27.62) lms_dfe_xx ()

Type	Function
Description	Least-Mean-Square Decision Feedback Equalizer. **lms_dfe_cc** : complex in/out **lms_dfe_ff** : float in/out
Usage	**gr.lms_dfe_xx(lambda_ff, lambda_fb, num_fftaps, num_fbtaps)**
Parameters	
Note	Needs more documentation

1.8.27.63) dpll_bb ()

Type	Function
Description	Detect the peak of a signal. If a peak is detected, this block outputs a 1, else it outputs 0's.
Usage	**gr.dpll_bb(period, gain)**
Parameters	**period** : ???????? **gain** : ???????
Note	Needs more documentation

1.8.27.64) pll_freqdet_cf ()

Type	Function
Description	Implements a PLL which locks to the input frequency and outputs an estimate of that frequency. Useful for FM Demod. input: stream of complex; output: stream of floats. 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. Alpha is the phase gain (first order, units of radians per radian) and beta is the frequency gain (second order, units of radians per sampl per radian)
Usage	**gr.pll_freqdet_cf(alpha, beta, max_freq, min_freq)**
Parameters	**alpha :** **beta :** **max_freq :** **min_freq** :
Note	Needs more documentation

1.8.27.65) pll_refout_cc ()

Type	Function
Description	Implements a PLL which locks to the input frequency and outputs a carrier. input: stream of complex; output: stream of 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. Alpha is the phase gain (first order, units of radians per radian) and beta is the frequency gain (second order, units of radians per sample per radian)
Usage	**gr.pll_refout_cc(alpha, beta, max_freq, min_freq)**
Parameters	**alpha :** **beta :** **max_freq :** **min_freq** :
Note	1) Needs more documentation 2) If alpha = x, it was suggested that beta = 0.25 * x * x
Example	See hfx2.py in apps

1.8.27.66) pll_carriertracking_cc()

Type	Function
Description	Implements a PLL which locks to the input frequency and outputs the input signal mixed with that carrier. input: stream of complex; output: stream of 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. Alpha is the phase gain (first order, units of radians per radian) and beta is the frequency gain (second order, units of radians per sample per radian)
Usage	**gr.pll_carriertracking_cc(alpha, beta, max_freq, min_freq)**
Parameters	**alpha :** **beta :** **max_freq :** **min_freq** :

Note	Needs more documentation
Sub Function 1	gr.pll_carriertracking_cc.lock_detector() : Return bool True or False
Sub Function 2	gr.pll_carriertracking_cc.set_lock_threshold(value) : Set threshold value
Sub Function 3	gr.pll_carriertracking_cc.squelch_enable(on) : Set /reset squelch

1.8.27.67) pn_correlator_cc ()

Type	Function
Description	PN code sequential search correlator. Receives complex baseband signal, outputs complex correlation against reference PN code, one sample per PN code period
Usage	**gr.pn_correlator_cc(degree, mask, seed)**
Parameters	**degree:** **mask :** **seed :**
Note	Needs more documentation

1.8.27.68) probe_signal_f ()

Type	Function
Description	Sink that allows a samples ruuning in stream to be grabbed from Python.
Usage	**gr.probe_signal_f()**
Parameters	
Note	Needs more documenation
Sub Function 1	gr.probe_signal_f.level() : Return probed signal level
Example	See radio.py in apps

1.8.27.69) probe_avg_mag_sqrd_xx ()

Type	Function
Description	Sink that allows a samples ruuning in stream to be grabbed from Python. It computes avg magnitude squared. 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. **probe_avg_mag_sqrd_c :** input: gr_complex **probe_avg_mag_sqrd_f :** input: float **probe_avg_mag_sqrd_cf :** input: gr_complex, output : gr_float
Usage	**gr.proble_avg_mag_sqrd_xx(threshold_db, alpha)**
Parameters	**threshold_db** : The threshold value in dB **alpha**: The gain value (float) of a moving average filter (Time Constant in sec)
Note	Needs more documenation
Sub Function 1	gr.probe_ avg_mag_sqrd_xx.level() : Return double represent the probed level
Sub Function 2	gr.probe_ avg_mag_sqrd_xx.thresholdl() : Return double represent block threshold
Sub Function 3	gr.probe_ avg_mag_sqrd_xx.unmuted() : Return bool True or False
Sub Function 4	gr.probe_ avg_mag_sqrd_xx.set_thresholdl() : Set threshold
Sub Function 5	gr.probe_ avg_mag_sqrd_xx.set_alpha() : Set alpha
Example	See receive-path.py in digital folder

1.8.27.70) ofdm_correlator ()

Type	Function
Description	Build an OFDM correlator and equalizer.Take a vector of complex constellation points in from an FFT and performs a correlation and equalization. blocks This block takes the output of an FFT of a received OFDM symbol and finds the start of a frame based on two known symbols. It also looks at the surrounding bins in the FFT output for the correlation in case there is a large frequency shift in the data. This block assumes that the fine frequency shift has already been corrected and that the samples fall in the middle of one FFT bin. It then uses one of those known symbols to estimate the channel response over all subcarriers and does a simple 1-tap equalization on all subcarriers. This corrects for the phase and amplitude distortion caused by the channel.
Usage	**gr.ofdm_correlator(occupied_carriers, fft_length, cplen, known_symbol1, known_symbol2, max_fft_shift_len)**
Parameters	***occupied_carriers*** The number of subcarriers with data in the received symbol ***fft_length*** The size of the FFT vector (occupied_carriers + unused carriers) ***known_symbol1*** A vector of complex numbers representing a known symbol at the start of a frame (usually a BPSK PN sequence) ***known_symbol2*** A vector of complex numbers representing a known symbol at the start of a frame after known_symbol1 (usually a BPSK PN sequence). Both of these start symbols are differentially correlated to compensate for phase changes between symbols. ***max_fft_shift_len*** Set's the maximum distance you can look between bins for correlation
Note	Needs more documenation
Sub Function 1	gr. ofdm_correlator.snr () : Return an estimate of the SNR of the channel.
Example	

1.8.27.71) ofdm_cyclic_prefixer ()

Type	Function
Description	Adds a cyclic prefix vector to an input size long ofdm symbol (vector) and converts vector to a stream output_size long.
Usage	**gr.ofdm_cyclic_prefixer(input_size, output_size)**
Parameters	**input_size**: ?????????? **output_size**: ????????????
Note	Needs more documenation
Example	

1.8.27.72) ofdm_bpsk_mapper ()

Type	Function
Description	Take a message in and map to a vector of complex constellation points suitable for IFFT input to be used in an ofdm modulator. Simple BPSK version.
Usage	**gr.ofdm_bpsk_mapper (msgq_limit, occupied_carriers, fft_length)**
Parameters	**msgq_limit**: maximum number of messages in message queue **occupied_carriers**: ???????? **fft_length** : FFT length
Note	Needs more documentation
Sub Function 1	gr.ofdm_bpsk_mapper.msgq() : Return a pointer to msg queue
Example	

1.8.27.73) ofdm_bpsk_demapper ()

Type	Function
Description	Take a vector of complex constellation points in from an FFT and demodulate to a stream of bits. Simple BPSK version.
Usage	**gr.ofdm_bpsk_demapper (occupied_carriers)**
Parameters	**occupied_carriers**: ????????
Note	Needs more documentation
Example	

1.8.27.74) ofdm_ mapper_bcv ()

Type	Function
Description	Take a stream of bytes in and map to a vector of complex constellation points suitable for IFFT input to be used in an ofdm modulator. Abstract class must be subclassed with specific mapping.
Usage	**gr.ofdm_mapper_bcv (constellation, msgq_limit, occupied_carriers, fft_length)**
Parameters	**constellation** : vector of complex data **msgq_limit**: maximum number of messages in message queue **occupied_carriers**:???????? **fft_length** : FFT length
Note	Needs more documentation
Sub Function 1	gr.ofdm_mapper_bcv.msgq() : Return a pointer to msg queue
Example	

1.8.27.75) ofdm_qpsk_mapper ()

Type	Function
Description	Take a message in and map to a vector of complex constellation points suitable for IFFT input to be used in an ofdm modulator. Simple QPSK version.
Usage	**gr.ofdm_qpsk_mapper (msgq_limit, occupied_carriers, fft_length)**
Parameters	**msgq_limit**: maximum number of messages in message queue **occupied_carriers**: ???????? **fft_length** : FFT length
Note	Needs more documentation
Sub Function 1	gr.ofdm_qpsk_mapper.msgq() : Return a pointer to msg queue
Example	

1.8.27.76) ofdm_qam_mapper ()

Type	Function
Description	Take a message in and map to a vector of complex constellation points suitable for IFFT input to be used in an ofdm modulator. Simple QAM version.
Usage	**gr.ofdm_qam_mapper (msgq_limit, occupied_carriers, fft_length, m)**
Parameters	**msgq_limit**: maximum number of messages in message queue **occupied_carriers** : ???????? **fft_length** : FFT length **m** : ??????????
Note	Needs more documentation
Sub Function 1	gr.ofdm_qam_mapper.msgq() : Return a pointer to msg queue
Example	

1.8.27.77) ofdm_frame_sink ()

Type	Function
Description	Takes an OFDM symbol in, demaps it into bits of 0's and 1's, packs them into packets, and sends to to a message queue sink. NOTE: The mod input parameter simply chooses a pre-defined demapper/slicer. Eventually, we want to be able to pass in a reference to an object to do the demapping and slicing for a given modulation type.
Usage	**gr.ofdm_frame_sink (sym_position, sym_value_out, target_queue, occupied_tones)**
Parameters	**sym_position** : vector of complex **sym_value_out** : vector of unsigned characters **target_queue** : point to message queue **occupied_tones** : Integer
Note	Needs more documentation
Example	

1.8.27.78) ofdm_insert_preamble ()

Type	Function
Description	Insert "pre-modulated" preamble symbols before each payload. Input 1: stream of vectors of gr_complex [fft_length]. These are the modulated symbols of the payload. Input 2: stream of char. The LSB indicates whether the corresponding symbol on input 1 is the first symbol of the payload or not. It's a 1 if the corresponding symbol is the first symbol; otherwise 0.This implies that there must be at least 1 symbol in the payload. Output 1: stream of vectors of gr_complex [fft_length] These include the preamble symbols and the payload symbols. Output 2: stream of char. The LSB indicates whether the corresponding symbol on input 1 is the first symbol of a packet (i.e., the first symbol of the preamble.) It's a 1 if the corresponding symbol is the first symbol, otherwise 0.
Usage	**gr.ofdm_insert_preamble(fft_length, preamble)**
Parameters	**fft_length** : FFT length **preamble** : vector of complex vectors
Note	Needs more documentation
Example	

1.8.27.79) ofdm_sampler ()

Type	Function
Description	Does the rest of the OFDM stuff ?????????????
Usage	**gr.ofdm_sampler(fft_length, symbol_length)**
Parameters	**fft_length** : FFT length **symbol_length**: ?????????????????
Note	Needs more documentation
Example	

1.8.27.80) regenerate_bb ()

Type	Function
Description	Make a regenerate block. Detect the peak of a signal and repeat every period samples. If a peak is detected, this block outputs a 1 repeated every period samples until reset by detection of another 1 on the input or stopped after max_regen regenerations have occurred. Note that if max_regen= (-1)/ULONG_MAX, then the regeneration will run forever.
Usage	**gr.regenerate_bb(period, max_regen)**
Parameters	***period*** : The number of samples between regenerations ***max_regen*** : The maximum number of regenerations to perform; if set to ULONG_MAX, it will regenerate continuously.
Note	Needs more documentation
Sub Function 1	gr.regenerate_bb.set_max_regen (regen) : Reset the maximum regeneration count; this will reset the current regen.
Sub Function 2	gr.regenerate_bb.set_period (period) : Reset the period of regenerations; this will reset the current regen.
Example	

1.8.27.81) costas_loop_cc ()

Type	Function
Description	A Costas loop carrier recovery module. Carrier tracking PLL for QPSK. Input: complex; output: complex .The Costas loop can have two output streams: stream 1 is the baseband I and Q; stream 2 is the normalized frequency of the loop order must be 2 or 4. The Costas loop locks to the center frequency of a signal and downconverts it to baseband. The second (order=2) order loop is used for BPSK where the real part of the output signal is the baseband BPSK signal and the imaginary part is the error signal. When order=4, it can be used for quadrature modulations where both I and Q (real and imaginary) are outputted. More details can be found online: J. Feigin, "Practical Costas loop design: Designing a simple and inexpensive BPSK Costas loop carrier recovery circuit," RF signal processing, pp. 20-36, 2002. http://rfdesign.com/images/archive/0102Feigin20.pdf
Usage	**gr.costas_loop_cc(alpha, beta, max_freq, min_freq, order)**
Parameters	***alpha*** : The loop gain used for phase adjustment ***beta*** : The loop gain for frequency adjustments ***max_freq***:The maximum frequency deviation (normalized frequency) the loop can handle ***min_freq*** : The minimum frequency deviation (normalized frequency) the loop can handle ***order*** : The loop order, either 2 or 4
Note	Needs more documentation
Example	

1.8.27.82) pa_2x2_phase_combiner ()

Type	Function
Description	pa_2x2 phase combiner. Anntenas are arranged like this: 2 3 0 1 dx and dy are lambda/2.
Usage	**gr.pa_2x2_phase_combiner()**
Parameters	
Note	Needs more documentation
Sub Function 1	gr.pa_2x2_phase_combiner.theta() : Return theta
Sub Function 2	gr.pa_2x2_phase_combiner.set_theta(theta) : Set theta (float)
Example	

1.8.27.83) kludge_copy ()

Type	Function
Description	output[i] = input[i] .This is a short term kludge to work around a problem with the hierarchical block impl.
Usage	**gr.kludge_copy(itemsize)**
Parameters	**itemsize**: One of gr.sizeof_short, gr.sizeof_gr_complex, gr.sizeof_float ,gr_sizeof_char
Note	
Example	

1.8.27.84) prefs ()

Type	Function
Description	Base class for representing user preferences in windows INI files. The real implementation is in Python, and is accessable from C++ via the magic of SWIG directors.
Usage	
Parameters	
Note	Needs more documentation
Example	

1.8.27.85) test ()

Type	Function
Description	Test class for testing runtime system (setting up buffers and such.). This block does not do any usefull actual data processing. It just exposes setting all standard block parameters using the contructor or public methods. This block can be usefull when testing the runtime system. You can force this block to have a large history, decimation factor and/or large output_multiple. The runtime system should detect this and create large enough buffers all through the signal chain.
Usage	
Parameters	
Note	Needs more documentation
Example	

1.8.27.86) unpack_k_bits_bb ()

Type	Function
Description	Converts a byte with k relevent bits to k output bytes with 1 bit in the LSB.
Usage	**gr.unpack_k_bits_bb(k)**
Parameters	
Note	Needs more documentation
Example	

1.8.27.87) correlate_access_code_bb ()

Type	Function
Description	Examine input for specified access code, one bit at a time. Input: stream of bits, 1 bit per input byte (data in LSB), output: stream of bits, 2 bits per output byte (data in LSB, flag in next higher bit). Each output byte contains two valid bits, the data bit, and the flag bit. The LSB (bit 0) is the data bit, and is the original input data, delayed 64 bits. Bit 1 is the flag bit and is 1 if the corresponding data bit is the first data bit following the access code. Otherwise the flag bit is 0.
Usage	**gr.correlate_access_code_bb(access_code, threshold)**
Parameters	***access_code*** : is string represented with 1 byte per bit, e.g., "010101010111000100" ***threshold*** : maximum number of bits that may be wrong
Note	Needs more documentation
Sub Function 1	gr.correlate_access_code_bb.set_access_code(access_code)
Example	

1.8.27.88) diff_phasor_cc ()

Type	Function
Description	???????????????????
Usage	
Parameters	
Note	Needs more documentation
Example	

1.8.27.89) constellation_decoder_cb ()

Type	Function
Description	???????????????????
Usage	**gr. constellation_decoder_cb(sym_position, sym_value_out)**
Parameters	**sym_position** : vector of complex **sym_value_out** : vector of unsigned charaters
Note	Needs more documentation
Sub Function 1	gr. constellation_decoder_cb.set_constellation(sym_position, sym_value_out)
Example	

1.8.27.90) binary_slicer_fb ()

Type	Function
Description	Slice float binary symbol outputting 1 bit output (the LSB of the output byte) per sample. If x <0 then output 0. If x >=0 then output 1
Usage	**gr. binary_slicer_fb()**
Parameters	
Note	
Example	

1.8.27.91) diff_encoder_bb ()

Type	Function
Description	Differential encoder.y[0] = (x[0] + y[-1]) % M
Usage	**gr. diff_encoder_bb(modulus)**
Parameters	
Note	
Example	

1.8.27.92) diff_decoder_bb ()

Type	Function
Description	Differential decoder.y[0] = (x[0] - x[-1]) % M
Usage	**gr. diff_decoder_bb(modulus)**
Parameters	
Note	
Example	

1.8.27.93) framer_sink_1 ()

Type	Function
Description	Given a stream of bits and access_code flags, assemble packets. Input: stream of bytes from gr_correlate_access_code_bb, output: none. Pushes assembled packet into target queue. The framer expects a fixed length header of 2 16-bit shorts containing the payload length, followed by the payload. If the 2 16-bit shorts are not identical, this packet is ignored. Better algs are welcome. The input data consists of bytes that have two bits used. Bit 0, the LSB, contains the data bit. Bit 1 if set, indicates that the corresponding bit is the the first bit of the packet. That is, this bit is the first one after the access code.
Usage	**gr. framer_sink_1(target_queue)**
Parameters	**target_queue** : pointer to message queue
Note	Needs more documenation
Example	

1.8.27.94) map_bb ()

Type	Function
Description	output[i] = map[input[i]]
Usage	**gr. map_bb(map)**
Parameters	**map** : a vector of intgers
Note	Needs more documenation
Example	

1.8.27.95) feval ()

Type	Function
Description	Base class for evaluating a function: void -> void This class is designed to be subclassed in Python or C++ and is callable from both places. It uses SWIG's "director" feature to implement the magic. It's slow. Don't use it in

	a performance critical path. Override eval to define the behavior. Use calleval to invoke eval (this kludge is required to allow a python specific "shim" to be inserted.
Usage	**gr. feval()**
Parameters	
Sub Function 1	ge.feval.calleval()
Note	Needs more documenation
Example	

1.8.27.96) feval_xx ()

Type	Function
Description	Base class for evaluating a function.This class is designed to be subclassed in Python or C++ and is callable from both places. It uses SWIG's "director" feature to implement the magic. It's slow. Don't use it in a performance critical path. Override eval to define the behavior. Use calleval to invoke eval (this kludge is required to allow a python specific "shim" to be inserted. **feval_cc** : complex to complex **feval_dd** : double to double **feval_ll** : long to long
Usage	**gr. feval_xx()**
Parameters	
Sub Function 1	ge.feval_xx.calleval()
Note	Needs more documenation
Example	

1.8.27.97) pwr_squelch_xx ()

Type	Function
Description	Gate or zero output when input power below threshold. pwr_squelch_cc : complex input, complex output pwr_squelch_ff : float input, float output
Usage	**gr. pwr_squelch_xx(db, alpha, ramp, gate)**
Parameters	**db** : threshold value **alpha**: The gain value (float) of a moving average filter (Time Constant in sec) **ramp** : integer represents rise/fail time im msec **gate** : bool True or False
Sub Function 1	gr.pwr_squelch_xx.threshold() : Return threshold
Sub Function 2	gr.pwr_squelch_xx.set_threshold(db) : Set threshold
Sub Function 3	gr.pwr_squelch_xx.set_alpha(alpha) : Set alpha
Sub Function 4	gr.pwr_squelch_xx.ramp() : Return ramp
Sub Function 5	gr.pwr_squelch_xx.set_ramp(ramp) : Set ramp
Sub Function 6	gr.pwr_squelch_xx.gate() : Return bool True or False
Sub Function 7	gr.pwr_squelch_xx.set_gate(on) : Set threshold
Sub Function 8	gr.pwr_squelch_xx.unmuted() : Return bool True or False
Note	Needs more documenation
Example	

1.8.27.98) squelch_base_xx ()

Type	Function
Description	?????????????????????????????? **squelch_base_cc** : complex input, complex output

	squelch_base_ff : float input, float output
Usage	**gr. squelch_base_xx(name, ramp, gate)**
Parameters	**name** : ????? **ramp** : integer represents rise/fail time im msec **gate** : bool True or False
Sub Function 1	gr.squelch_ base_xx.squelch_range() : Return range
Sub Function 2	gr. squelch_ base_xx.ramp() : Return ramp
Sub Function 3	gr. squelch_ base_xx.set_ramp(ramp) : Set ramp
Sub Function 4	gr. squelch_ base_xx.gate() : Return bool True or False
Sub Function 5	gr. squelch_ base_xx.set_gate(on) : Set threshold
Sub Function 6	gr. squelch_ base_xx.unmuted() : Return bool True or False
Note	Needs more documenation
Example	

1.8.27.99) ctcss_squelch_ff ()

Type	Function
Description	Gate or zero output if ctcss tone not present
Usage	**gr. ctcss_squelch_ff(rate, freq, level, len, ramp, gate)**
Parameters	**rate** : sampling rate **freq** : tone frequency **level** : tone level **ramp** : integer represents rise/fail time im msec **gate** : bool True or False
Sub Function 1	gr.ctcss_squelch_ff.level() : Return level
Sub Function 2	gr.ctcss_squelch_ff.set_level(level) : Set level
Sub Function 3	gr.ctcss_squelch_ff.len() : Return length
Sub Function 4	gr.ctcss_squelch_ff.squelch_range() : Return squelch range
Sub Function 5	gr.ctcss_squelch_ffr_amp() : Return ramp
Sub Function 6	gr.ctcss_squelch_ff.set_ramp(ramp) : Set ramp
Sub Function 7	gr.ctcss_squelch_ff.gate() : Return True or False
Sub Function 8	gr.ctcss_squelch_ff.set_gate(gate) : Set Gate
Sub Function 9	gr.ctcss_squelch_ff.unmuted() : Return True or False
Note	Needs more documenation
Example	

1.8.27.100) feedforward_agc_cc ()

Type	Function
Description	Non-causal AGC which computes required gain based on max absolute value over nsamples.
Usage	**gr. feedforward_agc_cc(nsamples, refrence)**
Parameters	**nsamples** : number of samples **refrence** : refrence value
Note	
Example	

1.8.27.101) bin_statistics_f ()

Type	Function
Description	Sink block that controls frequency scanning and record frequency domain statistics.
Usage	**gr. bin_statistics(vlen, msgq, tune, tune_delay, dwell_delay)**
Parameters	vlen : vector length (fft size) msgq : pointer to msg queue tune : python callback function (tune type is gr.feval_dd()) tune_delay : Time to delay (in number of samples) after changing frequency dwell_delay : Time to dwell (in number of samples) at a given frequncy
Note	Needs more documenation
Example	See usrp_spectrum_sense.py

1.8.27.102) glfsr_source_x ()

Type	Function
Description	**glfsr_source_f** : Galois LFSR pseudo-random source generating float outputs -1.0 - 1.0. **glfsr_source_b** : Galois LFSR pseudo-random source generating 0 or 1.
Usage	**gr. glfsr_source_x(degree, repeat, mask, seed)**
Parameters	**degree** : **repeat** : bool True or False **mask** : **seed** :
Sub Function 1	gr. glfsr_source_x.period() : Return period
Sub Function 2	gr. glfsr_source_x.mask() : Return mask
Note	Needs more documenation
Example	

1.9) gnuradio/ window.py

Type	Python file
Description	Routines for designing window functions for FFT.
Examples	
Note	

1.9.1) hamming()

Type	Function
Description	Design Hamming window
Usage	**window.hamming(fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.2) hanning()

Type	Function
Description	Design Hanning window
Usage	**window.hanning(fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.3) welch()

Type	Function
Description	Design Welch window
Usage	**window.welch(fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.4) parzen()

Type	Function
Description	Design Parzen window
Usage	**window.parzen(fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.5) bartlett()

Type	Function
Description	Design Bartlett window
Usage	**window.bartlett(fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.6) blackman2()

Type	Function
Description	Design Blackman2 window
Usage	**window.blackman2(fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.7) blackman3()

Type	Function
Description	Design Blackman3 window
Usage	**window.blackman3(fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.8) blackman4()

Type	Function
Description	Design Blackman4 window
Usage	**window.blackman4(fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.9) exponential()

Type	Function
Description	Design Exponential window
Usage	**window.exponential (fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.10) riemann()

Type	Function
Description	Design Riemann window
Usage	**window.riemann (fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.11) blackmanharris ()

Type	Function
Description	Design Blackmanharris window
Usage	**window.blackmanharris (fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.12) nuttall()

Type	Function
Description	Design Nuttall window
Usage	**window.nuttal (fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.9.13) kaiser()

Type	Function
Description	Design Kaiser window
Usage	**window.kaiser (fft_size)**
Parameters	**fft_size** : number of window taps
Examples	
Note	

1.10) gnuradio/ video_sdl.py

Type	Python file
Description	Simple Direct Media Layer (SDL) routines for displaying video.
Examples	
Note	

1.10.1) video_sdl_sink_uc()

Type	Function
Description	Video sink using SDL. Input signature is one, two or three streams of unsigned char. One stream: stream is grey (Y) two streems: first is grey (Y), second is alternating U and V Three streams: first is grey (Y), second is U, third is V Input samples must be in the range [0,255].
Usage	**video_sdl.video_sdl_sink_uc(framerate, width, height, format, dst_width, dst_height)**
Parameters	**framerate** : double **width** : integer **height** : integer **format** : unsigned integer **dst_width** : integer **dst_height** : integer
Examples	
Note	

1.10.2) video_sdl_sink_s()

Type	Function
Description	Video sink using SDL. Input signature is one, two or three streams of signed shorts. One stream: stream is grey (Y) two streems: first is grey (Y), second is alternating U and V Three streams: first is grey (Y), second is U, third is V Input samples must be in the range [0,255].
Usage	**video_sdl.video_sdl_sink_s(framerate, width, height, format, dst_width, dst_height)**
Parameters	**framerate** : double **width** : integer (640) **height** : integer (480) **format** : unsigned integer **dst_width** : integer **dst_height** : integer
Examples	
Note	

1.10.3) sink_uc()

Type	Function
Description	Video sink using SDL. Input signature is one, two or three streams of unsigned char. One stream: stream is grey (Y) two streems: first is grey (Y), second is alternating U and V Three streams: first is grey (Y), second is U, third is V Input samples must be in the range [0,255].
Usage	**video_sdl.sink_uc(framerate, width, height, format, dst_width, dst_height)**
Parameters	**framerate** : double **width** : integer **height** : integer **format** : unsigned integer **dst_width** : integer **dst_height** : integer
Examples	
Note	

1.10.4) sink_s()

Type	Function
Description	Video sink using SDL. Input signature is one, two or three streams of signed shorts. One stream: stream is grey (Y) two streems: first is grey (Y), second is alternating U and V Three streams: first is grey (Y), second is U, third is V Input samples must be in the range [0,255].
Usage	**video_sdl. sink_s(framerate, width, height, format, dst_width, dst_height)**
Parameters	**framerate** : double **width** : integer **height** : integer **format** : unsigned integer **dst_width** : integer **dst_height** : integer
Examples	
Note	

1.11) gnuradio/ trellis.py

Type	Python file
Description	Trellis coding ??????????????????
Examples	
Note	

1.11.1) fsm()

Type	Function
Description	Finite state machine ????????????
Usage	**trellis.fsm()**
Parameters	
Examples	
Note	Needs more documentation

1.11.2) interleaver()

Type	Function
Description	??????????
Usage	**trellis.interleaver()**
Parameters	
Examples	
Note	Needs more documentation

1.11.3) trellis_permutation ()

Type	Function
Description	Permutation ?????????????
Usage	**trellis.trellis_permutation()**
Parameters	
Examples	
Note	Needs more documentation

1.11.4) trellis_siso_f ()

Type	Function
Description	???????????????
Usage	**trellis.trellis_siso_f()**
Parameters	
Examples	
Note	Needs more documentation

1.11.5) trellis_encoder_xx ()

Type	Function
Description	??????????????? **trellis_encoder_bb** **trellis_encoder_bi** **trellis_encoder_bs** **trellis_encoder_ii** **trellis_encoder_si** **trellis_encoder_ss** **trellis_encoder_bb** **trellis_encoder_bb**
Usage	
Parameters	
Examples	
Note	Needs more documentation

1.11.6) trellis_metrics_x ()

Type	Function
Description	??????????????? **trellis_metrics_c** **trellis_metrics_f** **trellis_metrics_i** **trellis_metrics_s**
Usage	
Parameters	
Examples	
Note	Needs more documentation

1.11.7) trellis_viterbi_x ()

Type	Function
Description	??????????????? **trellis_viterbi_b** **trellis_viterbi_i** **trellis_viterbi_s**
Usage	
Parameters	
Examples	
Note	Needs more documentation

1.11.8) trellis_viterbi_combined_xx ()

Type	Function
Description	??????????????? **trellis_viterbi_combined_cb** **trellis_viterbi_combined_ci** **trellis_viterbi_combined_cs** **trellis_viterbi_combined_fb** **trellis_viterbi_combined_fi** **trellis_viterbi_combined_fs** **trellis_viterbi_combined_ib** **trellis_viterbi_combined_ii** **trellis_viterbi_combined_is** **trellis_viterbi_combined_sb** **trellis_viterbi_combined_si** **trellis_viterbi_combined_ss**
Usage	
Parameters	
Examples	
Note	Needs more documentation

1.12) gnuradio/ sounder.py

Type	Python file
Description	This is a work-in-progress implementation of a m-sequence based channel sounder for GNU Radio and the USRP. In typical use, the user would run the sounder as a transmitter on oneUSRP, and a receiver on another at a different location. The receiver will determine the impulse response of the RF channel in between.

The sounder uses a custom FPGA bitstream that is able to generate and receive a sounder waveform across a full 32 MHz wide swath of RF spectrum; the waveform generation and impulse response processing occur in logic in the USRP FPGA and not in the host PC. This avoids the USB throughput bottleneck entirely. Unfortunately, there is still roll-off in the AD9862 digital up-converter interpolation filter that impacts the outer 20% of bandwidth, but this can be compensated for by measuring and subtracting out this response during calibration.

The sounder is based on sending a maximal-length PN code modulated as BPSK with the supplied center frequency, with a chip-rate of 32 MHz. The receiver correlates the received signal across all phases of the PN code and outputs an impulse response vector. As auto-correlation of an m-sequence is near zero for any relative phase shift, the actual measured energy at a particular phase shift is related to the impulse response for that time delay. This is the same principle used in spread-spectrum RAKE receivers such as are used with GPS and CDMA.

The transmitter is designed to work only with the board in side A. The receiver may be in side A or side B. The boards may be standalone LFTX/LFRXs or RFX daughterboards.

To use, the following script is installed into $prefix/bin:

```
Usage: usrp_sounder.py [options]

Options:
  -h, --help            show this help message and exit
  -R RX_SUBDEV_SPEC, --rx-subdev-spec=RX_SUBDEV_SPEC
                        select USRP Rx side A or B
  -f FREQ, --frequency=FREQ
                        set frequency to FREQ in Hz, default is 0.0
  -d DEGREE, --degree=DEGREE
                        set sounding sequence degree (2-12), default is 12,
  -t, --transmit        enable sounding transmitter
  -r, --receive         enable sounding receiver
  -l, --loopback        enable digital loopback, default is disabled
  -v, --verbose         enable verbose output, default is disabled
  -D, --debug           enable debugging output, default is disabled
  -F FILENAME, --filename=FILENAME
                        log received impulse responses to file
```

To use with an LFTX board, set the center frequency to 16M:

```
$ usrp_sounder.py -f 16M -t
```

The sounder receiver command line is:

```
$ usrp_sounder.py -f 16M -r -F output.dat
```

You can vary the m-sequence degree between 2 and 12, which will create sequence lengths between 3 and 4095 (128 us). This will affect how frequently the receiver can calculate impulse response vectors.

The correlator uses an O(N^2) algorithm, by using an entire PN period of the received signal to correlate at each lag value. Thus, using a degree 12 PN code of length 4095, it takes 4095*4095/32e6 seconds to calculate a single impulse response vector, about a half a second. One can reduce this time by a factor of 4 for each decrement in PN code degree, but this also reduces the inherent processing gain by 6 dB as well.

The impulse response vectors are written to a file in complex float format, and consist of the actual impulse response with a noise floor dependent on the PN code degree in use.

There is a loopback test mode that causes the sounding waveform to be routed back to the receiver inside the USRP:

	$ usrp_sounder.py -r -t -l -F output.dat The resulting impulse response will be a spike followed by a near zero value for the rest of the period. Synchronization at the receiver is not yet implemented, so the actual impulse response may be time shifted an arbitrary value within the the impulse response vector. If one assumes the first to arrive signal is the strongest, then one can circularly rotate the vector until the peak is at time zero.
Examples	
Note	

1.12.1) sounder_tx ()

Type	Function
Description	Sounder_tx function
Usage	**sounder.sounder_tx (loopback=False, ampl=4096,verbose=False, debug=False)**
Parameters	
Examples	
Note	Needs more documentation

1.12.2) sounder_rx ()

Type	Function
Description	Sounder_rx function
Usage	**sounder.sounder_rx(subdev_spec=None,gain=None,length=1,alpha=1.0,msgq=None,loopback=False,verbose=False,debug=False)**
Parameters	
Examples	
Note	Needs more documentation

1.12.3) sounder ()

Type	Function
Description	
Usage	**sounder.sounder(transmit=False,receive=False,loopback=False,rx_subdev_spec=None,ampl=0x1FFF,frequency=0.0,rx_gain=None,degree=12,length=1,alpha=1.0,msgq=None,verbose=False,debug=False)**
Parameters	
Examples	
Note	Needs more documentation

1.13) gnuradio/ radar_mono.py

Type	Python file
Description	This GNU Radio component implements a monostatic radar transmitter and receiver. It uses a custom FPGA build to generate a linear FM chirp waveform directly in the USRP. Echo returns are recorded to a file for offline analysis. The LFM chirp can be up to 32 MHz in width, whose center frequency is set by which transmit daughter board is installed. This gives a range resolution of approximately 5 meters.
Examples	
Note	

1.13.1) radar_tx ()

Type	Function
Description	Transmitter object. Uses usrp_sink, but only for a handle to the FPGA registers.
Usage	**radar_mono.radar_tx(options)**
Parameters	
Examples	
Note	

1.13.2) radar_rx ()

Type	Function
Description	Receiver object. Uses usrp_source_c to receive echo records.
Usage	**radar_mono.radar_rx(options,callback)**
Parameters	
Examples	
Note	

1.13.3) radar ()

Type	Function
Description	???????
Usage	**radar_mono.radar(options,callback)**
Parameters	
Examples	
Note	

1.14) gnuradio/ ra.py

Type	Python file
Description	Radio astronomy. This file was automatically generated by SWIG
Examples	
Note	Needs more documentation

1.15) gnuradio/ packet_util.py

Type	Python file
Description	Utilities for packet handling
Examples	
Note	

1.15.1) conv_packed_binary_string_to_1_0_string ()

Type	Function
Description	'\xAF' --> '10101111'

Usage	**packet_util.conv_packed_binary_string_to_1_0_string(s)**
Parameters	**s**: string
Examples	
Note	

1.15.2) conv_1_0_string_to_packed_binary_string ()

Type	Function
Description	'10101111' -> ('\xAF', False) Basically the inverse of conv_packed_binary_string_to_1_0_string, but also returns a flag indicating if we had to pad with leading zeros to get to a multiple of 8.
Usage	**packet_util. conv_1_0_string_to_packed_binary_string (s)**
Parameters	**s**: string
Examples	
Note	

1.15.3) make_packet ()

Type	Function
Description	Build a packet, given access code, payload, and whitener offset. Packet will have access code at the beginning, followed by length, payload and finally CRC-32.
Usage	**packet_util. make_packet(payload, samples_per_symbol, bits_per_symbol, access_code=default_access_code, pad_for_usrp=True, whitener_offset=0, whitening=True)**
Parameters	**payload**: Packet payload, len [0, 4096] **samples_per_symbol**: samples per symbol (needed for padding calculation) type samples_per_symbol: int **bits_per_symbol**: (needed for padding calculation) type bits_per_symbol: int **access_code**: string of ascii 0's and 1's **pad_for_usrp**: If true, packets are padded such that they end up a multiple of 128 samples **whitener_offset :** offset into whitener string to use [0-16] **whitening**: Turn whitener on or off type whitening: bool
Examples	
Note	

1.15.4) unmake_packet ()

Type	Function
Description	Return (ok, payload)
Usage	**packet_util. unmake_packet(whitened_payload_with_crc, whitener_offset=0, dewhitening=True)**
Parameters	**whitened_payload_with_crc**: string **whitener_offset**: offset into whitener string to use [0-16] **dewhitening**: Turn whitener on or off type dewhitening: bool
Examples	
Note	

1.15.5) _npadding_bytes ()

Type	Function
Description	Generate sufficient padding such that each packet ultimately ends up being a multiple of 512 bytes when sent across the USB. We send 4-byte samples across the USB (16-bit I and 16-bit Q), thus we want to pad so that after modulation the resulting packet is a multiple of 128 samples. **Returns** number of bytes of padding to append.
Usage	**packet_util. _npadding_bytes(pkt_byte_len, samples_per_symbol, bits_per_symbol)**
Parameters	**ptk_byte_len**: len in bytes of packet, not including padding. **samples_per_symbol**: samples per bit (1 bit / symbolwidth GMSK) type samples_per_symbol: int **bits_per_symbol**: bits per symbol (log2(modulation order)) type bits_per_symbol: int
Examples	
Note	

1.16) gnuradio/ optfir.py

Type	Python file
Description	Routines for designing optimal FIR filters. For a great intro to how all this stuff works, see section 6.6 of "Digital Signal Processing: A Practical Approach", Emmanuael C. Ifeachor and Barrie W. Jervis, Adison-Wesley, 1993. ISBN 0-201-54413-X.
Examples	
Note	

1.16.1) low_pass ()

Type	Function
Description	Low pass filter design.
Usage	**optfir.low_pass (gain, Fs, freq1, freq2, passband_ripple_db, stopband_atten_db, nextra_taps=0)**
Parameters	
Examples	See ayfabtu.py
Note	Needs more documentation

1.16.2) high_pass ()

Type	Function
Description	High pass filter design.
Usage	**optfir.high_pass (Fs, freq1, freq2, stopband_atten_db, passband_ripple_db, nextra_taps=0)**
Parameters	
Examples	
Note	1) FIXME : The high_pass is broken 2) Needs more documentation

1.17) gnuradio/ ofdm_packet_util.py

Type	Python file
Description	Utilities for OFDM packet handling
Examples	
Note	

1.17.1) conv_packed_binary_string_to_1_0_string ()

Type	Function
Description	'\xAF' --> '10101111'
Usage	**ofdm_packet_util.conv_packed_binary_string_to_1_0_string(s)**
Parameters	**s**: string
Examples	
Note	

1.17.2) conv_1_0_string_to_packed_binary_string ()

Type	Function
Description	'10101111' -> ('\xAF', False) Basically the inverse of conv_packed_binary_string_to_1_0_string, but also returns a flag indicating if we had to pad with leading zeros to get to a multiple of 8.
Usage	**ofdm_packet_util. conv_1_0_string_to_packed_binary_string (s)**
Parameters	**s**: string
Examples	
Note	

1.17.3) make_packet ()

Type	Function
Description	Build a packet, given access code, payload, and whitener offset. Packet will have access code at the beginning, followed by length, payload and finally CRC-32.
Usage	**ofdm_packet_util. make_packet(payload, samples_per_symbol, bits_per_symbol, pad_for_usrp=True, whitener_offset=0, whitening=True)**
Parameters	**payload**: packet payload, len [0, 4096] **samples_per_symbol**: samples per symbol (needed for padding calculation) type samples_per_symbol: int **bits_per_symbol**: (needed for padding calculation) type bits_per_symbol: int **whitener_offset :** offset into whitener string to use [0-16) **pad_for_usrp**: If true, packets are padded such that they end up a multiple of 128 samples **whitening**: Turn whitener on or off type whitening: bool
Examples	
Note	

1.17.4) unmake_packet ()

Type	Function
Description	Return (ok, payload)
Usage	**ofdm_packet_util. unmake_packet(whitened_payload_with_crc, whitener_offset=0, dewhitening=True)**

Parameters	**whitened_payload_with_crc**: string **whitener_offset :** offset into whitener string to use [0-16) **dewhitening**: Turn whitener on or off type dewhitening: bool
Examples	
Note	

1.17.5) _npadding_bytes ()

Type	Function
Description	Generate sufficient padding such that each packet ultimately ends up being a multiple of 512 bytes when sent across the USB. We send 4-byte samples across the USB (16-bit I and 16-bit Q), thus we want to pad so that after modulation the resulting packet is a multiple of 128 samples. **Returns** number of bytes of padding to append.
Usage	**ofdm_packet_util. _npadding_bytes(pkt_byte_len, samples_per_symbol, bits_per_symbol)**
Parameters	**ptk_byte_len**: len in bytes of packet, not including padding. **samples_per_symbol**: samples per bit (1 bit / symbolwidth GMSK) type samples_per_symbol: int **bits_per_symbol**: bits per symbol (log2(modulation order)) type bits_per_symbol: int
Examples	
Note	

1.18) gnuradio/ modulation_utils.py

Type	Python file
Description	Miscellaneous utilities for managing modulations and demodulations, as well as other items useful in dealing with generalized handling of different modulations and demods.
Examples	
Note	

1.18.1) type_1_mods ()

Type	Function
Description	Type 1 modulators accept a stream of bytes on their input and produce complex baseband output
Usage	**modulation_utils.type_1_mods()**
Parameters	
Examples	See tunnel.py
Note	Needs more documentation

1.18.2) type_1_demods ()

Type	Function
Description	Type 1 demodulators accept complex baseband input and produce a stream of bits, packed1 bit / byte as their output. Their output is completely unambiguous. There is no Needsto resolve phase or polarity ambiguities.
Usage	**modulation_utils.type_1_demods()**

Parameters	
Examples	See tunnel.py
Note	Needs more documentation

1.19) gnuradio/ local_calibrator.py

Type	Python file
Description	Simple class for allowing local definition of a calibration function for raw samples coming from the RA detector chain. Each observatory is different, and rather than hacking up the main code in usrp_ra_receiver we define the appropriate function here. For example, one could calibrate the output in Janskys, rather than dB.
Examples	
Note	NO LONGER USED

1.20) gnuradio/gr_unittest.py

Type	Python file
Description	Add support for unit testing
Examples	
Note	

1.21) gnuradio/eng_option.py

Type	Python file
Description	Add support for engineering notation to optparse.OptionParser
Examples	
Note	

1.22) gnuradio/eng_notation.py

Type	Python file
Description	Change engineering notation (example 5e-9 <==> 5n)
Examples	
Note	

1.22.1) num_to_str ()

Type	Function
Description	Convert a number to a string in engineering notation. E.g., 5e-9 -> 5n
Usage	**eng_notation.num_to_str(n)**
Parameters	
Examples	
Note	

1.22.2) str_to_num ()

Type	Function
Description	Convert a string in engineering notation to a number. E.g., '15m' -> 15e-3
Usage	**eng_notation.str_to_num(value)**
Parameters	
Examples	
Note	

1.23) gnuradio/audio_oss.py

Type	Python file
Description	Open Sound System (oss) sound interface for audio sink and source.This file was automatically generated by SWIG
Examples	
Note	

1.24) gnuradio/audio_alsa.py

Type	Python file
Description	Advanced Linux Sound Architecture (ALSA) sound interface for audio sink and source.This file was automatically generated by SWIG
Examples	
Note	

1.25) gnuradio/audio.py

Type	Python file
Description	This is the 'generic' audio or soundcard interface. known_modules = ('audio_alsa', 'audio_oss', 'audio_osx', 'audio_jack', 'audio_portaudio'). The behavior of this module is controlled by the [audio] audio_module configuration parameter. If it is 'auto' we attempt to import modules from the known_modules list, using the first one imported successfully. If [audio] audio_module is not 'auto', we assume it's the name of an audio module and attempt to import it.
Examples	
Note	

1.25.1) source ()

Type	Function
Description	Audio source. Output signature is one or two streams of floats. Output samples will be in the range [-1,1].
Usage	**audio.source(sampling_rate, device_name=””, ok_to_block=true)**
Parameters	**sampling_rate** : integer **device_name** : string **ok_to_block** : bool
Examples	
Note	

1.25.2) sink ()

Type	Function
Description	Audio sink. Input signature is one or two streams of floats. Input samples must be in the range [-1, 1].
Usage	**audio.sink(sampling_rate, device_name="", ok_to_block=true)**
Parameters	**sampling_rate** : integer **device_name** : string **ok_to_block** : bool
Examples	`sink = audio.sink(sample_rate, "plughw:0,0")`
Note	

1.26) gnuradio/atsc.py

Type	Python file
Description	Support for ATSC signal handling. This file was automatically generated by SWIG.
Examples	
Note	Needs more documentation

1.27) gnuradio/usrp.py

Type	Python file
Description	Configuration interface for the USRP
Examples	
Note	

1.27.1) source_x()

Type	Function
Description	interface to Universal Software Radio Peripheral Rx path **source_c()** : complex data source **source_s()** : short interleaved data source
Usage	**usrp.source_x(which=0, decim_rate=64, nchan=1, mux=0x32103210, mode=0, fusb_block_size=0, fusb_nblocks=0, fpga_filename="", firmware_filename="")**
Parameters	
Examples	
Note	

1.27.1.1) tune()

Type	Sub Function
Description	Set the center frequency we're interested in. Tuning is a two step process. First we ask the front-end to tune as close to the desired frequency as it can. Then we use the result of that operation and our target_frequency to determine the value for the digital down converter. **Returns** False if failure else tune_result
Usage	**usrp.source_x.tune(u, chan, subdev, target_freq)**
Parameters	**u**: instance of usrp.source_* **chan**: DDC number

	type chan: int **subdev**: daughterboard subdevice **target_freq**: frequency in Hz
Examples	
Note	

1.27.1.2) has_rx_halfband()

Type	Sub Function
Description	To check if the FPGA implement a final Rx half-band filter? If it doesn't, the maximum decimation factor with proper gain is 1/2 of what it would otherwise be.
Usage	**usrp.source_x.has_rx_halfband()**
Parameters	
Examples	
Note	

1.27.1.3) nddcs()

Type	Sub Function
Description	Return number of Digital Down Converters implemented in FPGA, this will be 0, 1, 2, or 4.
Usage	**usrp.source_x.nddcs()**
Parameters	
Examples	
Note	

1.27.2) sink_x()

Type	Function
Description	Interface to Universal Software Radio Peripheral Tx path **sink_c()** : complex data source **sink _s()** : short interleaved data source
Usage	**usrp.sink _x(which=0, interp_rate=128, nchan=1, mux=0x98,** **fusb_block_size=0, fusb_nblocks=0,** **fpga_filename="", firmware_filename="")**
Parameters	
Examples	
Note	

1.27.2.1) tune()

Type	Sub Function
Description	Set the center frequency we're interested in. Tuning is a two step process. First we ask the front-end to tune as close to the desired frequency as it can. Then we use the result of that operation and our target_frequency to determine the value for the digital down converter. Returns False if failure else tune_result
Usage	**usrp.sink_x.tune(u, chan, subdev, target_freq)**
Parameters	**u**: instance of usrp.sink_* **chan**: DUC number

	type chan: int **subdev**: daughterboard subdevice **target_freq**: frequency in Hz
Examples	
Note	Needs more documentation

1.27.2.2) has_tx_halfband()

Type	Sub Function
Description	To check if the FPGA implement a final Tx half-band filter?
Usage	**usrp.sink_x.has_tx_halfband()**
Parameters	
Examples	
Note	

1.27.2.3) nducs()

Type	Sub Function
Description	Return number of Digital up Converters implemented in FPGA, this will be 0,1,or 2,.
Usage	**usrp.sink_x.nducs()**
Parameters	
Examples	
Note	

1.27.3) determine_rx_mux_value ()

Type	Function
Description	A utility to determine appropriate Rx mux value as a function of the subdevice choosen and thecharacteristics of the respective daughterboard. **Returns**: the Rx mux value Figure out which A/D's to connect to the DDC. Each daughterboard consists of 1 or 2 subdevices. (At this time, all but the Basic Rx have a single subdevice. The Basic Rx has two independent channels, treated as separate subdevices). subdevice 0 of a daughterboard may use 1 or 2 A/D's. We determine this by checking the is_quadrature() method. If subdevice 0 uses only a single A/D, it's possible that the daughterboard has a second subdevice, subdevice 1, and it uses the second A/D. If the card uses only a single A/D, we wire a zero into the DDC Q input. (side, 0) says connect only the A/D's used by subdevice 0 to the DDC. (side, 1) says connect only the A/D's used by subdevice 1 to the DDC.
Usage	**usrp.determine_rx_mux_value(u, subdev_spec)**
Parameters	**u**: Instance of USRP source **subdev_spec**: Tuple represent (side, subdev). type subdev_spec: (side, subdev), where side is 0 or 1 and subdev is 0 or 1
Examples	
Note	

1.27.4) tune_result ()

Type	Function
Description	Container for intermediate tuning information. **Return** tunning informations
Usage	tune_result.**baseband_freq** : Return the resultant baseband frequency tune_result **.dxc_freq** : Return the used DDC or DUC frequency tune_result.**residual_freq** : Return the residual frequency after tunning tune_result **.inverted** : Return True if the spectrum is inverted, otherwise return False
Parameters	
Examples	
Note	

1.27.5) determine_tx_mux_value ()

Type	Function
Description	A utility to determine the appropriate Tx mux value as a function of the subdevice choosen. **Returns**: The Tx mux value This is simpler than the rx case. Either you want to talk to side A or side B. If you want to talk to both sides at once, determine the value manually.
Usage	**usrp.determine_tx_mux_value(u, subdev_spec):**
Parameters	**u**: instance of USRP sink **subdev_spec**: Tuple represent (side, subdev). . type subdev_spec: (side, subdev), where side is 0 or 1 and subdev is 0
Examples	
Note	

1.27.6) selected_subdev ()

Type	Function
Description	A utility to return the user specified daughterboard subdevice. **Returns**: An instance derived from db_base.
Usage	**usrp.selected_subdev (u, subdev_spec)**
Parameters	**u**: Instance of USRP sink or source **subdev_spec**: Tuple represent (side, subdev) type subdev_spec: (side, subdev), where side is 0 or 1 and subdev is 0 or 1
Examples	
Note	

1.27.7) calc_dxc_freq ()

Type	Function
Description	A utility to calculate the frequency to be used for setting the digital up or down converter. **Return**: 2-tuple (ddc_freq, inverted) where ddc_freq is the value for the ddc and inverted is True if we're operating in an inverted Nyquist zone
Usage	**usrp.calc_dxc_freq(target_freq, baseband_freq, fs)**

Parameters	**target_freq**: desired RF frequency (Hz) type target_freq: number **baseband_freq**: The RF frequency that corresponds to DC in the IF. type baseband_freq: number **fs**: converter sample rate type fs: number
Examples	
Note	

1.27.8) pick_rx_subdevice()

Type	Function
Description	If the user didn't specify an rx subdevice on the command line, try for one of these, in order: FLEX_400, FLEX_900, FLEX_1200, FLEX_1800, FLEX_2400, TV_RX, DBS_RX, and BASIC_RX, whatever's on **side A**. **Return** a subdev_spec
Usage	**usrp.pick_rx_subdevice(u)**
Parameters	**u**: Instance of USRP source
Examples	
Note	

1.27.9) pick_tx_subdevice()

Type	Function
Description	If the user didn't specify a tx subdevice on the command line, try for one of these, in order: FLEX_400, FLEX_900, FLEX_1200, FLEX_1800, FLEX_2400, BASIC_TX, whatever's **on side A**. **Return** a subdev_spec
Usage	**usrp.pick_rx_subdevice(u)**
Parameters	**u**: Instance of USRP source
Examples	
Note	

1.27.10) pick_ subdev()

Type	Function
Description	Pick whatever in side A **Return:** subdev specification
Usage	**usrp.pick_subdev(u, candidates)**
Parameters	**u**: usrp instance sink or source **candidates**: list of usrp dbids which are : usrp_dbid.BASIC_TX = 0x0000 usrp_dbid.BASIC_RX = 0x0001 usrp_dbid.DBS_RX = 0x0002 usrp_dbid.TV_RX = 0x0003 usrp_dbid.FLEX_400_RX = 0x0004 usrp_dbid.FLEX_900_RX = 0x0005 usrp_dbid.FLEX_1200_RX = 0x0006 usrp_dbid.FLEX_2400_RX = 0x0007 usrp_dbid.FLEX_400_TX = 0x0008 usrp_dbid.FLEX_900_TX = 0x0009 usrp_dbid.FLEX_1200_TX = 0x000a usrp_dbid.FLEX_2400_TX = 0x000b usrp_dbid.TV_RX_REV_2 = 0x000c usrp_dbid.DBS_RX_REV_2_1 = 0x000d

	usrp_dbid.LF_TX = 0x000e usrp_dbid.LF_RX = 0x000f usrp_dbid.FLEX_400_RX_MIMO_A = 0x0014 usrp_dbid.FLEX_900_RX_MIMO_A = 0x0015 usrp_dbid.FLEX_1200_RX_MIMO_A = 0x0016 usrp_dbid.FLEX_2400_RX_MIMO_A = 0x0017 usrp_dbid.FLEX_400_TX_MIMO_A = 0x0018 usrp_dbid.FLEX_900_TX_MIMO_A = 0x0019 usrp_dbid.FLEX_1200_TX_MIMO_A = 0x001a usrp_dbid.FLEX_2400_TX_MIMO_A = 0x001b usrp_dbid.FLEX_400_RX_MIMO_B = 0x0024 usrp_dbid.FLEX_900_RX_MIMO_B = 0x0025 usrp_dbid.FLEX_1200_RX_MIMO_B = 0x0026 usrp_dbid.FLEX_2400_RX_MIMO_B = 0x0027 usrp_dbid.FLEX_400_TX_MIMO_B = 0x0028 usrp_dbid.FLEX_900_TX_MIMO_B = 0x0029 usrp_dbid.FLEX_1200_TX_MIMO_B = 0x002a usrp_dbid.FLEX_2400_TX_MIMO_B = 0x002b usrp_dbid.FLEX_1800_RX = 0x0030 usrp_dbid.FLEX_1800_TX = 0x0031 usrp_dbid.FLEX_1800_RX_MIMO_A = 0x0032 usrp_dbid.FLEX_1800_TX_MIMO_A = 0x0033 usrp_dbid.FLEX_1800_RX_MIMO_B = 0x0034 usrp_dbid.FLEX_1800_TX_MIMO_B = 0x0035 usrp_dbid.TV_RX_REV_3 = 0x0040 usrp_dbid.WBX_LO_TX = 0x0050 usrp_dbid.WBX_LO_RX = 0x0051
Examples	
Note	

1.28) gnuradio/usrp1.py

Type	Python file
Description	Configuration interface for the USRP Rev 1 and later
Examples	
Note	

1.28.1) source_x()

Type	Function
Description	Interface to Universal Software Radio Peripheral Rx path **source_c()** : complex data source **source_s()** : short interleaved data source
Usage	**usrp.source_x(which=0, decim_rate=64, nchan=1, mux=0x32103210, mode=0, fusb_block_size=0, fusb_nblocks=0, fpga_filename="", firmware_filename="")**
Parameters	
Examples	
Note	

1.28.1.1) set_decim_rate()

Type	Sub Function
Description	Set decimator rate. Rate must be **EVEN** and in **[8, 256].** The final complex sample rate across the USB is adc_freq () / decim_rate () *nchannels()

Usage	**usrp.source_x.set_decim_rate(rate)**
Parameters	**rate** : unsigned integer represents the decimation rate
Examples	
Note	

1.28.1.2) set_nchannels()

Type	Sub Function
Description	Set number of active ddc channels. Nchannels must be 1, 2, 3 or 4.
Usage	**usrp.source_x.set_nchannels(nchan)**
Parameters	**nchan** : integer
Examples	
Note	

1.28.1.3) set_mux()

<table>
<tr><td>Type</td><td>Sub Function</td></tr>
<tr><td>Description</td><td>This determines which ADC (or constant zero) is connected to each DDC input. There are 4 DDCs. Each has two inputs.
Mux value:

  3 2 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-------+-------+-------+-------+-------+-------+-------+-------+
| Q3 | I3 | Q2 | I2 | Q1 | I1 | Q0 | I0 |
+-------+-------+-------+-------+-------+-------+-------+-------+

Each 4-bit I field is either 0,1,2,3
Each 4-bit Q field is either 0,1,2,3 or 0xf (input is const zero)
All Q's must be 0xf or none of them may be 0xf</td></tr>
<tr><td>Usage</td><td>usrp.source_x.set_mux(mux)</td></tr>
<tr><td>Parameters</td><td>mux : integer</td></tr>
<tr><td>Examples</td><td></td></tr>
<tr><td>Note</td><td>To specify an integer value using hex format using gru.hexint() function</td></tr>
</table>

1.28.1.4) set_rx_freq()

Type	Sub Function
Description	Set the center frequency of the digital down converter. Channel must be in the range [0,3]. freq is the center frequency in Hz. freq may be either negative or postive. The frequency specified is quantized. Use rx_freq to retrieve the actual value used.
Usage	usrp.source_x.set_rx_freq(channel,freq)
Parameters	***channel**:* which ddc channel [0, 3] ***freq*** : double, the frequency
Examples	
Note	

1.28.1.5) set_ddc_phase()

Type	Sub Function
Description	Set the digital down converter phase register.
Usage	**usrp.source_x.set_ddc_phase(channel,phase)**
Parameters	***channel**:* which ddc channel [0, 3] ***phase*** : 32-bit integer phase value.

Examples	
Note	

1.28.1.6) set_fpga_mode()

Type	Sub Function
Description	Set fpga special modes
Usage	**usrp.source_x.set_fpga_mode(mode)**
Parameters	***mode*** *: one of* *FPGA_MODE_NORMAL , FPGA_MODE_LOOPBACK, FPGA_MODE_COUNTING, FPGA_MODE_COUNTING_32BIT*
Examples	
Note	

1.28.1.7) set_verbose()

Type	Sub Function
Description	Print usrp configuration
Usage	**usrp.source_x.set_verbose(verbose)**
Parameters	***verbose*** *: bool true or false*
Examples	
Note	

1.28.1.8) set_pga()

Type	Sub Function
Description	Set A/D Programmable Gain Amplifier (PGA). Gain is rounded to closest setting supported by hardware. **Return** true if sucessful
Usage	**usrp.source_x.set_pga(which, gain_in_db)**
Parameters	***which*** : which A/D [0,3] ***gain_in_db*** : double gain value (linear in dB) in range [0.0,20.0]
Examples	
Note	

1.28.1.9) pga()

Type	Sub Function
Description	Return programmable gain amplifier gain setting in dB.
Usage	**usrp.source_x.pga (which)**
Parameters	***which*** : which A/D [0,3]
Examples	
Note	

1.28.1.10) pga_min()

Type	Sub Function
Description	**Return** minimum legal PGA setting in dB.

Usage	**usrp.source_x.pga_min()**
Parameters	
Examples	
Note	

1.28.1.11) pga_max()

Type	Sub Function
Description	**Return** maximum legal PGA setting in dB.
Usage	**usrp.source_x.pga_max()**
Parameters	
Examples	
Note	

1.28.1.12) pga_db_per_step()

Type	Sub Function
Description	**Return** hardware step size of PGA (linear in dB).
Usage	**usrp.source_x.pga_db_per_step()**
Parameters	
Examples	
Note	

1.28.1.13) fpga_master_clock_freq()

Type	Sub Function
Description	**Return** fpga master clock frequency
Usage	**usrp.source_x.fpga_master_clock_freq()**
Parameters	
Examples	
Note	

1.28.1.14) converter_rate()

Type	Sub Function
Description	**Return** A/D converter rate
Usage	**usrp.source_x.converter_rate()**
Parameters	
Examples	
Note	

1.28.1.15) decim_rate()

Type	Sub Function
Description	**Return** decimation rate
Usage	**usrp.source_x.decim_rate()**
Parameters	
Examples	

Note	

1.28.1.16) nchannels()

Type	Sub Function
Description	**Return** number of active ddc channels
Usage	**usrp.source_x.nchannels()**
Parameters	
Examples	
Note	

1.28.1.17) mux()

Type	Sub Function
Description	**Return** mux setting
Usage	**usrp.source_x.mux()**
Parameters	
Examples	
Note	

1.28.1.18) rx_freq()

Type	Sub Function
Description	**Return** channels ddc frequency
Usage	**usrp.source_x.rx_freq(channel)**
Parameters	**channel** : which ddc channel [0,3]
Examples	
Note	

1.28.1.19) daughterboard_id()

Type	Sub Function
Description	**Return** daughterboard ID for given Rx daughterboard slot. Slot A =0, Slot B=1. daughterboard id >= 0 if successful -1 if no daugherboard -2 if invalid EEPROM on daughterboard
Usage	**usrp.source_x.daughterboard_id(which_dboardl)**
Parameters	**which_dboard** : Which slot o or 1
Examples	
Note	

1.28.1.20) write_aux_dac()

Type	Sub Function
Description	Write auxiliary digital to analog converter.
Usage	**usrp.source_x. write_aux_dac (*which_dboard*, *which_dac*, *value*)**
Parameters	***which_dboard***: [0,1] which slot , SLOT_TX_A and SLOT_RX_A share the same AUX

	DAC's. SLOT_TX_B and SLOT_RX_B share the same AUX DAC's. ***which_dac***: [2,3] TX slots must use only 2 and 3. ***value*** : in range of [0,4095]
Examples	
Note	

1.28.1.21) read_aux_dac()

Type	Sub Function
Description	Read auxiliary analog to digital converter. **Return** value in the range [0, 4095] if successful, else READ_FAILED.
Usage	**usrp.source_x. read_aux_dac (*which_dboard*, *which_adc*)**
Parameters	***which_dboard*** : [0,1] which slot ***which_adc*** : [0,1]
Examples	
Note	

1.28.1.22) write_eeprom()

Type	Sub Function
Description	Write EEPROM on motherboard or any daughterboard. **Return** true if successful
Usage	**usrp.source_x. write_eeprom(i2c_addr, eeprom_offset, buf)**
Parameters	***i2c_addr*** : I2C bus address of EEPROM ***eeprom_offset*** : byte offset in EEPROM to begin writing ***buf*** : the data to write
Examples	
Note	

1.28.1.23) read_eeprom()

Type	Sub Function
Description	Read bytes from EEPROM on motherboard or any daughterboard. **Return** the data read if successful, else a zero length string.
Usage	**usrp.source_x. read_eeprom(i2c_addr, eeprom_offset, len)**
Parameters	***i2c_addr*** : I2C bus address of EEPROM ***eeprom_offset*** : byte offset in EEPROM to begin reading ***buf*** : number of bytes to read
Examples	
Note	

1.28.1.24) write_i2c()

Type	Sub Function
Description	Write to I2C peripheral. Writes are limited to a maximum of of 64 bytes. **Return** true ff successful.
Usage	**usrp.source_x. write_i2c(i2c_addr, buf)**
Parameters	***i2c_addr*** : I2C bus address (7-bits) ***buf*** : The data to write

Examples	
Note	

1.28.1.25) read_i2c()

Type	Sub Function
Description	Read from I2C peripheral. Reads are limited to a maximum of of 64 bytes. **Return** the data read if successful, else a zero length string.
Usage	**usrp.source_x.read_i2c(i2c_addr, len)**
Parameters	***i2c_addr*** : I2C bus address (7-bits) ***len*** *: number of bytes to read*
Examples	
Note	

1.28.1.26) set_adc_offset()

Type	Sub Function
Description	Set ADC offset correction.
Usage	**usrp.source_x.set_adc_offset(which, offset)**
Parameters	***which*** : which ADC[0,3]: ***offset*****:** 16-bit value to subtract from raw ADC input.
Examples	
Note	

1.28.1.27) set_dac_offset()

Type	Sub Function
Description	Set DAC offset correction.
Usage	**usrp.source_x.set_dac_offset(which, offset, offset_pin)**
Parameters	***which*** : which DAC[0,3] ***offset***: 10-bit offset value (ambiguous format: See AD9862 datasheet). ***offset_pin***: 1-bit value. If 0 offset applied to -ve differential pin; If 1 offset applied to +ve differential pin.
Examples	
Note	

1.28.1.28) set_adc_buffer_bypass()

Type	Sub Function
Description	Control ADC input buffer.
Usage	**usrp.source_x.set_adc_buffer_bypass(which, bypass)**
Parameters	***which*** : which ADC [0,3] ***bypass*** *:* if non-zero, bypass input buffer and connect input directly to switched cap SHA input of RxPGA.
Examples	
Note	

1.28.1.29) serial_number()

Type	Sub Function
Description	**Return** the usrp's serial number. Return non-zero length string iff successful.
Usage	**usrp.source_x.serial_number()**
Parameters	
Examples	
Note	

1.28.1.30) _write_oe()

Type	Sub Function
Description	Write direction register (output enables) for pins that go to daughterboard. Each d'board has 16-bits of general purpose i/o. Setting the bit makes it an output from the FPGA to the d'board.This register is initialized based on a value stored in the d'board EEPROM. In general, you shouldn't be using this routine without a very good reason. Using this method incorrectly will kill your USRP motherboard and/or daughterboard.
Usage	**usrp.source_x._write_oe(which_dboard, value, mask)**
Parameters	***which_dboard*** : [0,1] which d'board ***value*** : value to write into register ***mask*** : which bits of value to write into reg
Examples	
Note	

1.28.1.31) write_io()

Type	Sub Function
Description	Write daughterboard i/o pin value.
Usage	**usrp.source_x.write_io(which_dboard, value, mask)**
Parameters	***which_dboard*** : [0,1] which d'board ***value*** : value to write into register ***mask*** : which bits of value to write into reg
Examples	
Note	

1.28.1.32) read_io()

Type	Sub Function
Description	Read daughterboard i/o pin value. **Return** register value if successful, else READ_FAILED
Usage	**usrp.source_x.read_io(which_dboard)**
Parameters	***which_dboard*** : [0,1] which d'board
Examples	
Note	

1.28.1.33) set_dc_offset_cl_enable()

Type	Sub Function
Description	Enable/disable automatic DC offset removal control loop in FPGA. If the corresponding bit is set, enable the automatic DC offset correction control loop. The 4 low bits are significant: ADC0 = (1 << 0) ADC1 = (1 << 1) ADC2 = (1 << 2) ADC3 = (1 << 3) By default the control loop is enabled on all ADC's.
Usage	**usrp.source_x.set_dc_offset_cl_enable(bits, mask)**
Parameters	***bits*** : which control loops to enable ***mask*** : which bits to pay attention to
Examples	
Note	

1.28.1.34) set_format()

Type	Sub Function
Description	Specify Rx data format. Rx data format control register 3 2 1 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 +---+-+-+---------+-------+ | Reserved (Must be zero) |B|Q| WIDTH | SHIFT | +---+-+-+---------+-------+ SHIFT specifies arithmetic right shift [0, 15] WIDTH specifies bit-width of I & Q samples across the USB [1, 16] (not all valid) Q if set deliver both I & Q, else just I B if set bypass half-band filter. Right now the acceptable values are: B Q WIDTH SHIFT 0 1 16 0 0 1 8 8 More valid combos to come. Default value is 0x00000300 16-bits, 0 shift, deliver both I & Q.
Usage	**usrp.source_x.set_format(format)**
Parameters	***format*** : unsigned integer format specifier
Examples	See usrp_fft.py
Note	

1.28.1.35) make_format()

Type	Sub Function
Description	Make data format.
Usage	**usrp.source_x.makeformat(width=16, shift=0, want_q=true,bypass_halfband=false)**
Parameters	**width** : integer **shift** : integer **want_q** : bool true or false, Do you want data Q channel or only the I Channel?. **bypass_halfband** : bool true or false
Examples	See usrp_fft.py
Note	

1.28.1.36) format()

Type	Sub Function
Description	**Return** current data format.
Usage	**usrp.source_x.format()**
Parameters	
Examples	
Note	

1.28.1.37) format_width()

Type	Sub Function
Description	Retuen format data width
Usage	**usrp.source_x.format_width(format)**
Parameters	
Examples	
Note	

1.28.1.38) format_shift()

Type	Sub Function
Description	Return format data shift
Usage	**usrp.source_x.format_shift(format)**
Parameters	
Examples	
Note	

1.28.1.39) format_want_q()

Type	Sub Function
Description	Return format want_q. Do you want Q samples?!
Usage	**usrp.source_x.format_want_q(format)**
Parameters	
Examples	
Note	

1.28.1.40) format_bypass_halfband()

Type	Sub Function
Description	Retuen format bypass halfband filter
Usage	**usrp.source_x.format_bypass_halfband(format)**
Parameters	
Examples	
Note	

1.28.1.41) _write_fpga_reg()

Type	Sub Function
Description	Write FPGA register. **Return** True if successful
Usage	**usrp.source_x._write_fpga_reg(regno,value)**
Parameters	***regno*** : 7-bit register number

	value : 32-bit value
Examples	
Note	

1.28.1.42) _write_fpga_reg_masked()

Type	Sub Function
Description	Write FPGA register masked. **Return** True if successful
Usage	**usrp.source_x._write_fpga_reg_masked(regno, value, mask)**
Parameters	***regno*** : 7-bit register number ***value*** : 16-bit value **mask** : 16 bit mask
Examples	
Note	

1.28.1.43) _read_fpga_reg()

Type	Sub Function
Description	Read FPGA registers. Return register value if successful, else READ_FAILED
Usage	**usrp.source_x._read_fpga_reg(regno)**
Parameters	***regno*** : 7-bit register number
Examples	
Note	READ_FAILED = -99999

1.28.1.44) _write_9862()

Type	Sub Function
Description	Write AD9862 register. **Return** true if successful
Usage	**usrp.source_x._write_9862(which_codec, regno, value)**
Parameters	***which_codec :*** 0 or 1 ***regno*** : 6-bit register number ***value*** : 8-bit value
Examples	
Note	

1.28.1.45) _read_9862()

Type	Sub Function
Description	Read AD9862 registers. **Return** register value if successful, else READ_FAILED
Usage	**usrp.source_x._read_9862(which_codec, regno)**
Parameters	***which_codec*** : 0 or 1

	regno : 6-bit register number
Examples	
Note	

1.28.1.46) _write_spi()

Type	Sub Function
Description	Write data to SPI bus peripheral. SPI == "Serial Port Interface". SPI is a 3 wire bus plus a separate enable for each peripheral. The common lines are SCLK,SDI and SDO. The FX2 always drives SCLK and SDI, the clock and data lines from the FX2 to the peripheral. When enabled, a peripheral may drive SDO, the data line from the peripheral to the FX2. The SPI_READ and SPI_WRITE commands are formatted identically. Each specifies which peripherals to enable, whether the bits should be transmistted Most Significant Bit first or Least Significant Bit first, the number of bytes in the optional header, and the number of bytes to read or write in the body. The body is limited to 64 bytes. The optional header may contain 0, 1 or 2 bytes. For an SPI_WRITE, the header bytes are transmitted to the peripheral followed by the the body bytes. For an SPI_READ, the header bytes are transmitted to the peripheral, then len bytes are read back from the peripheral.(see : usrp_spi_defs.h file). If format specifies that optional_header bytes are present, they are written to the peripheral immediately prior to writing buf. **Return** true if successful. Writes are limited to a maximum of 64 bytes.
Usage	**usrp.source_x._write_spi(optional_header, enables, format, buf)**
Parameters	***optional_header***: 0,1 or 2 bytes to write before buf. ***enables***: bitmask of peripherals to write. ***format*****:** transaction format. SPI_FMT_* ***buf*** : the data to write
Examples	
Note	

1.28.1.47) _read_spi()

Type	Sub Function
Description	Read data from SPI bus peripheral. **Return** the data read if successful, else a zero length string.
Usage	**usrp.source_x._read_spi(optional_header, enables, format, len)**
Parameters	***optional_header*** : 0,1 or 2 bytes to write before buf. ***enables*** : bitmask of peripherals to write. ***format*** : transaction format. SPI_FMT_* ***len*** : number of bytes to read.
Examples	
Note	

1.28.2) sink_x()

Type	Function
Description	interface to Universal Software Radio Peripheral Rx path **sink_c()** : complex data source **sink_s()** : short interleaved data source

Usage	**usrp.sink_x(which=0, interp_rate=128, nchan=1, mux=0x98, fusb_block_size=0, fusb_nblocks=0, fpga_filename="", firmware_filename="")**
Parameters	
Examples	
Note	

1.28.2.1) set_interp_rate()

Type	Sub Function
Description	Set interpolator rate. Rate must be in **[4, 512]** and a **multiple of 4**. The final complex sample rate across the USB is dac_freq () / interp_rate () * nchannels ()
Usage	**usrp.sink_x.set_interp_rate(rate)**
Parameters	**rate** : unsigned integer
Examples	
Note	

1.28.2.2) set_nchannels()

Type	Sub Function
Description	Set number of active duc channels, nchannels must be 1 or 2.
Usage	**usrp.sink_x.set_nchannels(nchan)**
Parameters	**nchan** : integer
Examples	
Note	

1.28.2.3) set_mux()

Type	Sub Function

Description	This determines which DAC is connected to each DUC input. There are 2 DUCs. Each has two inputs.
	
	Mux value:
	
	`  3                   2                   1`
	`1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0`
	`+-------------------------------+-------+-------+-------+-------+`
	`|                               | DAC3  | DAC2  | DAC1  |  DAC0 |`
	`+-------------------------------+-------+-------+-------+-------+`
	
	There are two interpolators with complex inputs and outputs.
	There are four DACs.
	
	Each 4-bit DACx field specifies the source for the DAC and whether or not that DAC is enabled. Each subfield is coded like this:
	
	`  3 2 1 0`
	`  +-+-----+`
	`  |E|  N  |`
	`  +-+-----+`
	
	Where E is set if the DAC is enabled, and N specifies which interpolator output is connected to this DAC.
	
	N which interp output

	--- ------------------- 0 chan 0 I 1 chan 0 Q 2 chan 1 I 3 chan 1 Q
Usage	**usrp.sink_x.set_mux(mux)**
Parameters	**mux** : integer
Examples	
Note	

1.28.2.4) set_tx_freq()

Type	Sub Function
Description	Set the frequency of the digital up converter, channel must be 0,1and freq is the center frequency in Hz. It must be in the range [-44M, 44M]. The frequency specified is quantized. Use tx_freq to retrieve the actual value used.
Usage	**usrp.sink_x.set_tx_freq(channel,freq)**
Parameters	***channel***: which duc channel [0, 1] ***freq*** : double
Examples	
Note	

1.28.2.5) set_verbose()

Type	Sub Function
Description	Print usrp configuration
Usage	**usrp.sink_x.set_verbose(verbose)**
Parameters	***verbose** : bool true or false*
Examples	
Note	

1.28.2.6) set_pga()

Type	Sub Function
Description	Set D/A Programmable Gain Amplifier (PGA). Gain is rounded to closest setting supported by hardware. Note that DAC 0 and DAC 1 share a gain setting as do DAC 2 and DAC 3. Setting DAC 0 affects DAC 1 and vice versa. Same with DAC 2 and DAC 3. Return true if sucessful
Usage	**usrp.sink_x.set_pga(which, gain_in_db)**
Parameters	***which*** : which D/A [0,3] ***gain_in_db*** : double gain value (linear in dB) in range [0.0,20.0]
Examples	
Note	

1.28.2.7) pga()

Type	Sub Function
Description	**Return** programmable gain amplifier gain setting in dB.

Usage	**usrp.sink_x.pga(which)**
Parameters	***which*** : which D/A [0,3]
Examples	
Note	

1.28.2.8) pga_min()

Type	Sub Function
Description	**Return** minimum legal PGA setting in dB.
Usage	**usrp.sink_x.pga_min()**
Parameters	
Examples	
Note	

1.28.2.9) pga_max()

Type	Sub Function
Description	**Return** maximum legal PGA setting in dB.
Usage	**usrp.sink_x.pga_max()**
Parameters	
Examples	
Note	

1.28.2.10) pga_db_per_step()

Type	Sub Function
Description	**Return** hardware step size of PGA (linear in dB).
Usage	**usrp.sink_x.pga_db_per_step()**
Parameters	
Examples	
Note	

1.28.2.11) fpga_master_clock_freq()

Type	Sub Function
Description	**Return** fpga master clock frequency
Usage	**usrp.sink_x.fpga_master_clock_freq()**
Parameters	
Examples	
Note	

1.28.2.12) converter_rate()

Type	Sub Function
Description	**Return** D/A converter rate
Usage	**usrp.sink_x.converter_rate()**
Parameters	

Examples	
Note	

1.28.2.13) interp_rate()

Type	Sub Function
Description	**Return** interpolation rate
Usage	**usrp.sink_x.interp_rate()**
Parameters	
Examples	
Note	

1.28.2.14) nchannels()

Type	Sub Function
Description	**Return** number of active duc channels
Usage	**usrp.sink_x.nchannels()**
Parameters	
Examples	
Note	

1.28.2.15) mux()

Type	Sub Function
Description	**Return** mux setting
Usage	**usrp.sink_x.mux()**
Parameters	
Examples	
Note	

1.28.2.16) tx_freq()

Type	Sub Function
Description	**Return** channels duc frequency
Usage	**usrp.sink_x.tx_freq(channel)**
Parameters	**channel** : which duc channel [0,3]
Examples	
Note	

1.28.2.17) daughterboard_id()

Type	Sub Function
Description	**Return** daughterboard ID for given Rx daughterboard slot. Slot A =0, Slot B=1. daughterboard id >= 0 if successful -1 if no daugherboard -2 if invalid EEPROM on daughterboard
Usage	**usrp.sink_x.daughterboard_id(which_dboardl)**
Parameters	**which_dboard** : o or 1, Slot number
Examples	
Note	

1.28.2.18) write_aux_dac()

Type	Sub Function
Description	Write auxiliary digital to analog converter.
Usage	**usrp.sink_x. write_aux_dac (*which_dboard*, *which_dac*, *value*)**
Parameters	***which_dboard*** : [0,1] which slot , SLOT_TX_A and SLOT_RX_A share the same AUX DAC's. SLOT_TX_B and SLOT_RX_B share the same AUX DAC's. ***which_dac*** : [2,3] TX slots must use only 2 and 3. ***value*** : in range of [0,4095]
Examples	
Note	

1.28.2.19) read_aux_dac()

Type	Sub Function
Description	Read auxiliary analog to digital converter. **Return** value in the range [0,4095] if successful, else READ_FAILED.
Usage	**usrp.sink_x. read_aux_dac (*which_dboard*, *which_adc*)**
Parameters	***which_dboard*** : [0,1] which slot ***which_adc*** : [0,1]
Examples	
Note	

1.28.2.20) write_eeprom()

Type	Sub Function
Description	Write EEPROM on motherboard or any daughterboard. **Return** true if successful
Usage	**usrp.sink_x. write_eeprom(i2c_addr, eeprom_offset, buf)**
Parameters	***i2c_addr*** : I2C bus address of EEPROM ***eeprom_offset*** : byte offset in EEPROM to begin writing ***buf*** : the data to write
Examples	
Note	

1.28.2.21) read_eeprom()

Type	Sub Function
Description	Read bytes from EEPROM on motherboard or any daughterboard. **Return** the data read if successful, else a zero length string.
Usage	**usrp.sink_x. read_eeprom(i2c_addr, eeprom_offset, len)**
Parameters	***i2c_addr*** : I2C bus address of EEPROM ***eeprom_offset*** : byte offset in EEPROM to begin reading ***buf*** : number of bytes to read
Examples	
Note	

1.28.2.22) write_i2c()

Type	Sub Function
Description	Write to I2C peripheral. **Return** true ff successful. Writes are limited to a maximum of of 64 bytes.
Usage	**usrp.sink_x. write_i2c(i2c_addr, buf)**
Parameters	***i2c_addr*** : I2C bus address (7-bits) ***buf*** : the data to write
Examples	
Note	

1.28.2.23) read_i2c()

Type	Sub Function
Description	Read from I2C peripheral. **Return** the data read if successful, else a zero length string. Reads are limited to a maximum of of 64 bytes.
Usage	**usrp.sink_x.read_i2c(i2c_addr, len)**
Parameters	***i2c_addr*** : I2C bus address (7-bits) ***len*** *: number of bytes to read*
Examples	
Note	

1.28.2.24) set_adc_offset()

Type	Sub Function
Description	Set ADC offset correction.
Usage	**usrp.sink_x.set_adc_offset(which, offset)**
Parameters	***which*** : which ADC[0,3]: ***offset*** : 16-bit value to subtract from raw ADC input.
Examples	
Note	

1.28.2.25) set_dac_offset()

Type	Sub Function
Description	Set DAC offset correction.
Usage	**usrp.sink_x.set_dac_offset(which, offset, offset_pin)**
Parameters	***which*** : which DAC[0,3]

	offset : 10-bit offset value (ambiguous format: See AD9862 datasheet). ***offset_pin*** : 1-bit value. If 0 offset applied to -ve differential pin; If 1 offset applied to +ve differential pin.
Examples	
Note	

1.28.2.26) set_adc_buffer_bypass()

Type	Sub Function
Description	Control ADC input buffer.
Usage	**usrp.sink_x.set_adc_buffer_bypass(which, bypass)**
Parameters	***which*** : which ADC [0,3] ***bypass :*** if non-zero, bypass input buffer and connect input directly to switched cap SHA input of RxPGA.
Examples	
Note	

1.28.2.27) serial_number()

Type	Sub Function
Description	**Return** the usrp's serial number. Return non-zero length string iff successful.
Usage	**usrp.sink_x.serial_number()**
Parameters	
Examples	
Note	

1.28.2.28) _write_oe()

Type	Sub Function
Description	Write direction register (output enables) for pins that go to daughterboard. Each d'board has 16-bits of general purpose i/o. Setting the bit makes it an output from the FPGA to the d'board.This register is initialized based on a value stored in the d'board EEPROM. In general, you shouldn't be using this routine without a very good reason. Using this method incorrectly will kill your USRP motherboard and/or daughterboard.
Usage	**usrp.sink_x._write_oe(which_dboard, value, mask)**
Parameters	***which_dboard*** : [0,1] which d'board ***value*** : value to write into register ***mask*** : which bits of value to write into reg
Examples	
Note	

1.28.2.29) write_io()

Type	Sub Function
Description	Write daughterboard i/o pin value.
Usage	**usrp.sink_x.write_io(which_dboard, value, mask)**
Parameters	***which_dboard*** : [0,1] which d'board

	value : value to write into register ***mask*** : which bits of value to write into reg
Examples	
Note	

1.28.2.30) read_io()

Type	Sub Function
Description	Read daughterboard i/o pin value. **Return** register value if successful, else READ_FAILED
Usage	**usrp.sink_x.read_io(which_dboard)**
Parameters	***which_dboard*** : [0,1] which d'board
Examples	
Note	READ_FAILED =-99999

1.28.2.31) _write_fpga_reg()

Type	Sub Function
Description	Write FPGA register. **Return** True if successful
Usage	**usrp.sink_x._write_fpga_reg(regno,value)**
Parameters	***regno*** : 7-bit register number ***value*** : 32-bit value
Examples	
Note	

1.28.2.32) _write_fpga_reg_masked()

Type	Sub Function
Description	Write FPGA register masked. **Return** True if successful
Usage	**usrp.sink_x._write_fpga_reg_masked(regno, value, mask)**
Parameters	***regno*** : 7-bit register number ***value*** : 16-bit value **mask** : 16 bit mask
Examples	
Note	

1.28.2.33) _read_fpga_reg()

Type	Sub Function
Description	Read FPGA registers. **Return** register value if successful, else READ_FAILED
Usage	**usrp.sink_x._read_fpga_reg(regno)**

Parameters	***regno*** : 7-bit register number
Examples	
Note	

1.28.2.34) _write_9862()

Type	Sub Function
Description	Write AD9862 registers. **Return** true if successful
Usage	**usrp.sink_x._write_9862(which_codec, regno, value)**
Parameters	***which_codec :*** 0 or 1 ***regno*** : 6-bit register number ***value*** : 8-bit value
Examples	
Note	

1.28.2.35) _read_9862()

Type	Sub Function
Description	Read AD9862 registers. **Return** register value if successful, else READ_FAILED
Usage	**usrp.sink_x._read_9862(which_codec, regno)**
Parameters	***which_codec*** : 0 or 1 ***regno*** : 6-bit register number
Examples	
Note	

1.28.2.36) _write_spi()

Type	Sub Function
Description	Write data to SPI bus peripheral. SPI == "Serial Port Interface". SPI is a 3 wire bus plus a separate enable for each peripheral. The common lines are SCLK,SDI and SDO. The FX2 always drives SCLK and SDI, the clock and data lines from the FX2 to the peripheral. When enabled, a peripheral may drive SDO, the data line from the peripheral to the FX2. The SPI_READ and SPI_WRITE commands are formatted identically. Each specifies which peripherals to enable, whether the bits should be transmistted Most Significant Bit first or Least Significant Bit first, the number of bytes in the optional header, and the number of bytes to read or write in the body. The body is limited to 64 bytes. The optional header may contain 0, 1 or 2 bytes. For an SPI_WRITE, the header bytes are transmitted to the peripheral followed by the the body bytes. For an SPI_READ, the header bytes are transmitted to the peripheral, then len bytes are read back from the peripheral.(see : usrp_spi_defs.h file). If format specifies that optional_header bytes are present, they are written to the peripheral immediately prior to writing buf. **Return** true if successful. Writes are limited to a maximum of 64 bytes.
Usage	**usrp.sink_x._write_spi(optional_header, enables, format, buf)**
Parameters	***optional_header*** : 0,1 or 2 bytes to write before buf. ***enables*** : bitmask of peripherals to write. ***format*** : transaction format. SPI_FMT_* ***buf*** : the data to write
Examples	
Note	

1.28.2.37) _read_spi()

Type	Sub Function
Description	Read data from SPI bus peripheral. **Return** the data read if successful, else a zero length string.
Usage	**usrp.sink_x._read_spi(optional_header, enables, format, len)**
Parameters	***optional_header*** : 0,1 or 2 bytes to write before buf. ***enables*** : bitmask of peripherals to write. ***format*** : transaction format. SPI_FMT_* ***len*** : number of bytes to read.
Examples	
Note	

1.29) gnuradio/usrp_multi.py

Type	Python file
Description	With this code you can connect two or more usrps (with a locked clock) and get synchronised samples.You must connect a (flat)cable between a dboard on the master in RXA and a dboard on the slave in RXA.You then put one usrp in master mode, put the other in slave mode. Warning, allways FIRST enable the slave before you enable the master This is to be sure you don't have two masters connecting to each other Otherwise you could ruin your hardware because the two sync outputs would be connected together. You determine which is the master by master_serialno (this is a text string a hexadecimal number).If you enter a serial number which is not found it will print the serial numbers which are available.If you give no serial number (master_serialno=None), the code will pick a Master for you.
Examples	The gnuradio-examples/python/multi_usrp directory contains examples
Note	To Synchroniza master and slave clocks, connect the 64MHz clocks between the boards with a short sma coax cable.(See the wiki on how to enable clock-out and clock-in http://gnuradio.org/trac/wiki/USRPClockingNotes) You Needsone board with a clock out and one board with a clock in. You can choose any of the two boards as master or slave; this is not dependant on which board has the clock-out or in. In the experiments, we had fewer problems when the board that has the clock-in will be the master board. You can use a standard 16-pole flatcable to connect tvrx, basic-rx or dbsrx boards. Of this 16pin flatcable only two pins are used (io15 and ground) For all new daughterboards which use up a lot of io pins you have to use a cable with fewer connections. The savest is using a 2pin headercable connected to io15,gnd (a cable like the ones used to connect frontpanel leds to the mainboard of a PC) If using basic rx board: Connect a 16-pole flatcable from J25 on basicrx/dbs_rx in rxa of the master usrp to J25 on basicrx/dbsrx in RXA of the slave usrp Don't twist the cable (Make sure the pin1 marker (red line on the flatcable) is on the same side of the connector (at io-8 on the master and at io8 on the slave.)) For basic_rx this means the marker should be on the side of the dboard with the sma connectors. For dbs_rx this means the marker should be on the side of the dboard with the two little chips. In other words, don't twist the cable; you will burn your board if you do. You can also connect a flatcable with multiple connectors from master-J25 to slave1-J25 to slave2-J25 to ... You will however have to think of something to create a common 64Mhz clock for more then two usrps. For all other daughterboards, connect a 2wire cable from masterRXA J25 io15, gnd to slaveRXA J25 io15, gnd. Now the hardware is setup.

1.29.1) multi_source_align ()

Type	Function
Description	Align multiple sources (USRPs) using sample numbers in the first channel. Takes two ore more sources producing interleaved shorts. **Produces**: nchan * nsources gr_complex output streams.
Usage	**usrp_multi.multi_source_align(fg, master_serialno,decim,nchan=2,pga_gain=0.0,cordic_freq=0.0,mux=None,align_interval=-1)**
Parameters	**nchan**: number of interleaved channels in source 2 or 4. **align_interval**: number of samples to minimally skip between alignments. default = -1 which means align only once per work call. **master_serial_no**: Serial_no of the usrp which should be the MASTER
Examples	
Note	

1.29.1.1) get_master_usrp ()

Type	Sub Function
Description	Get the instance of the master USRP
Usage	**usrp_multi.multi_source_align.get_master_usrp()**
Parameters	
Examples	
Note	

1.29.1.2) get_slave_usrp ()

Type	Sub Function
Description	Get the instance of the slave USRP
Usage	**usrp_multi.multi_source_align.get_slave_usrp()**
Parameters	
Examples	
Note	

1.29.1.3) get_master_source_c ()

Type	Sub Function
Description	Get the instance of the master USRP source channels. When we connect this instance, we can use the port number to get the channels from the master one (usrp_multi.multi_source_align.get_master_source_c(),1), (usrp_multi.multi_source_align.get_master_source_c(),2),
Usage	**usrp_multi.multi_source_align.get_master_source_c()**
Parameters	
Examples	
Note	These blocks have multiple outputs. output 0 is the sample counter (high bits in I, low bits in Q) You normally don't Needsthe samplecounters so you can ignore output 0 output 1 is the first aligend output channel (if you enable 2 or 4 channels) output 2 is the second output channel (only if you enable 4 channels)

1.29.1.4) get_slave_source_c ()

Type	Sub Function
Description	Get the instance of the slave USRP source channels. When we connect this instance, we can use the port number to get the channels from the slave one (usrp_multi.multi_source_align.get_slave_source_c(),1), (usrp_multi.multi_source_align.get_slave_source_c(),2),
Usage	**usrp_multi.multi_source_align.get_slave_source_c()**
Parameters	
Examples	
Note	These blocks have multiple outputs. output 0 is the sample counter (high bits in I, low bits in Q) You normally don't Needsthe samplecounters so you can ignore output 0 output 1 is the first aligend output channel (if you enable 2 or 4 channels) output 2 is the second output channel (only if you enable 4 channels)

1.29.1.5) sync ()

Type	Sub Function
Description	Called to synchroniza master and slave USRPs. You must call sync() at least once AFTER the flowgraph has started running.(This will synchronise the streams of the two usrps)
Usage	**usrp_multi.multi_source_align.sync()**
Parameters	
Examples	
Note	

1.29.1.6) print_db_info ()

Type	Sub Function
Description	Print master and slave daughterboards side and name
Usage	**usrp_multi.multi_source_align.print_db_info()**
Parameters	
Examples	
Note	

1.29.1.7) tune_all_rx ()

Type	Sub Function
Description	Tune all master and slave USRP 4 receive daughterboards to certen frequency.
Usage	**usrp_multi.multi_source_align.tune_all_rx(target_freq)**
Parameters	
Examples	
Note	This will only work reliably when you have all the same daughterboards.Otherwise set all freqs and gains individually.

1.29.1.8) set_gain_all_rx ()

Type	Sub Function
Description	Set the gain for all master and slave USRP 4 receive daughterboards.
Usage	**usrp_multi.multi_source_align. set_gain_all_rx(gain)**
Parameters	
Examples	
Note	This will only work reliably when you have all the same daughterboards.Otherwise set all freqs and gains individually.

1.30) gnuradio/tx_debug_gui.py

Type	Python file
Description	Debug tool for Tx. Transmit debugger.
usage	**`tx_debug_gui.tx_debug_gui(tx_subdev, title="Tx Debug")`**
Examples	
Note	To show the frame : debugger = tx_debug_gui.tx_debug_gui(subdev) debugger.Show(True)

1.31) gnuradio/flexrf_debug_gui.py

Type	Python file
Description	Debug tool for flexrf boards.
usage	**`flexrf_debug_gui.flexrf_debug_gui(flexrf, title="Flexrf Debug")`**
Parameters	**flexrf**: USRP source instance
Examples	
Note	To show the frame : debugger = flexrf_debug_gui.flexrf_debug_gui(flexrf) debugger.Show(True)

1.32) gnuradio/db_base.py

Type	Python file
Description	This is the abstract base class for all daughterboards.This defines the required operations and interfaces for all d'boards.
Note	All functions in this file will be mapped into daughterboards details as seen below.

1.33) gnuradio/db_basic.py

Type	Python file
Description	Handler for Basic Tx, Basic Rx, Low frequency TX, and Low frequency RX daughterboards
Note	

1.33.1) db_basic_tx ()

Type	Function
Description	Handler for Basic Tx daughterboards
Usage	**db_basic.db_basic_tx(usrp,which)**
Parameters	**usrp**: instance of usrp.sink **which**: which side: 0 or 1 corresponding to TX_A or TX_B respectively
Examples	
Note	Board Technical specifications : Min gain : 0 dB Max gain :20 dB Gain steps : 0.08 dB Min frequency : -1e09 Hz Max frequency : 1e09 Hz Frequency Step : 1e-6 Hz

1.33.1.1) dbid ()

Type	Sub Function
Description	**Return** daughter board ID
Usage	**db_basic.db_basic_tx.dbid()**
Parameters	
Examples	
Note	

1.33.1.2) name ()

Type	Sub Function
Description	**Return** daughter board name
Usage	**db_basic.db_basic_tx.name()**
Parameters	
Examples	
Note	

1.33.1.3) side_and_name ()

Type	Sub Function
Description	**Return** daughter board side and name
Usage	**db_basic.db_basic_tx.side_and_name()**
Parameters	
Examples	
Note	

1.33.1.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this d'board. Returns (min_freq, max_freq, step_size), return type is tuple.
Usage	**db_basic.db_basic_tx.freq_range()**
Parameters	
Examples	
Note	

1.33.1.5) set_freq ()

Type	Sub Function
Description	Set the frequency. Returns (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**db_basic.db_basic_tx.set_freq(target_freq)**
Parameters	**taget_freq**: target RF frequency in Hz type target_freq: float
Examples	
Note	

1.33.1.6) gain_range ()

Type	Sub Function
Description	**Return** range of gain that can be set by this d'board. Returns (min_gain, max_gain, step_size), where gains are expressed in decibels
Usage	**db_basic.db_basic_tx.gain_range()**
Parameters	
Examples	
Note	

1.33.1.7) set_gain ()

Type	Sub Function
Description	Set the gain. **Returns** True/False
Usage	**db_basic.db_basic_tx.set_gain(gain)**
Parameters	***gain***: gain in decibels
Examples	
Note	

1.33.1.8) is_quadrature ()

Type	Sub Function
Description	**Return** True if this daughterboard does quadrature up or down conversion. That is, return True if this board requires both I & Q analog channels. For this board, return True
Usage	**db_basic.db_basic_tx.is_quadrature()**

Parameters	
Examples	
Note	

1.33.2) db_basic_rx ()

Type	Function
Description	Handler for Basic Rx daughterboards
Usage	**db_basic.db_basic_rx(usrp,which,subdev)**
Parameters	**usrp**: instance of usrp.source **which**: which side: 0 or 1 corresponding to RX_A or RX_B respectively **subdev**: which analog i/o channel: 0 or 1
Examples	
Note	Board Technical specifications : Min gain : 0 dB Max gain :20 dB Gain steps : 1 dB Min frequency : -1e09 Hz Max frequency : 1e09 Hz Frequency Step : 1e-6 Hz

1.33.2.1) dbid ()

Type	Sub Function
Description	**Return** daughter board ID
Usage	**db_basic.db_basic_rx.dbid()**
Parameters	
Examples	
Note	

1.33.2.2) name ()

Type	Sub Function
Description	**Return** daughter board name
Usage	**db_basic.db_basic_rtx.name()**
Parameters	
Examples	
Note	

1.33.2.3) side_and_name ()

Type	Sub Function
Description	**Return** daughter board side and name
Usage	**db_basic.db_basic_rx.side_and_name()**
Parameters	
Examples	
Note	

1.33.2.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this d'board. Returns (min_freq, max_freq, step_size), return type tuple.
Usage	**db_basic.db_basic_rx.freq_range()**
Parameters	
Examples	
Note	

1.33.2.5) set_freq ()

Type	Sub Function
Description	Set the frequency. **Returns** (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**db_basic.db_basic_rx.set_freq(target_freq)**
Parameters	**target_freq**: target RF frequency in Hz type target_freq: float
Examples	
Note	

1.33.2.6) gain_range ()

Type	Sub Function
Description	**Return** range of gain that can be set by this d'board. Returns (min_gain, max_gain, step_size), where gains are expressed in decibels
Usage	**db_basic.db_basic_rx.gain_range()**
Parameters	
Examples	
Note	

1.33.2.7) set_gain ()

Type	Sub Function
Description	Set the gain. Returns True/False if successful.
Usage	**db_basic.db_basic_rx.set_gain(gain)**
Parameters	***gain***: gain in decibels
Examples	
Note	

1.33.2.8) is_quadrature ()

Type	Sub Function
Description	**Return** True if this daughterboard does quadrature up or down conversion. That is, return True if this board requires both I & Q analog channels. For this board, return False.
Usage	**db_basic.db_basic_rx.is_quadrature()**
Parameters	
Examples	
Note	

1.33.3) db_lf_rx ()

Type	Function
Description	Handler for Low frequency Rx daughterboards
Usage	**db_basic.db_lf_rx(usrp,which,subdev)**
Parameters	**usrp**: instance of usrp.source **which**: which side: 0 or 1 corresponding to RX_A or RX_B respectively **subdev**: which analog i/o channel: 0 or 1
Examples	
Note	Board Technical specifications : Min gain : 0 dB Max gain :20 dB Gain steps : 1 dB Min frequency : -32e06 Hz Max frequency : 32e06 Hz Frequency Step : 1e-6 Hz

1.33.3.1) dbid ()

Type	Sub Function
Description	**Return** daughter board ID
Usage	**db_basic.db_ lf _rx.dbid()**
Parameters	
Examples	
Note	

1.33.3.2) name ()

Type	Sub Function
Description	**Return** daughter board name
Usage	**db_basic.db_ lf _rtx.name()**
Parameters	
Examples	
Note	

1.33.3.3) side_and_name ()

Type	Sub Function
Description	**Return** daughter board side and name
Usage	**db_basic.db_ lf _rx.side_and_name()**

Parameters	
Examples	
Note	

1.33.3.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this d'board. Returns (min_freq, max_freq, step_size), return type tuple. We cover the first nyquist zone only
Usage	**db_basic.db_ lf _rx.freq_range()**
Parameters	
Examples	
Note	

1.33.3.5) set_freq ()

Type	Sub Function
Description	Set the frequency. **Returns** (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**db_basic.db_ lf _rx.set_freq(target_freq)**
Parameters	**target_freq**: target RF frequency in Hz type freq: float
Examples	
Note	

1.33.3.6) gain_range ()

Type	Sub Function
Description	**Return** range of gain that can be set by this d'board. returns (min_gain, max_gain, step_size), where gains are expressed in decibels
Usage	**db_basic.db_ lf _rx.gain_range()**
Parameters	
Examples	
Note	

1.33.3.7) set_gain ()

Type	Sub Function
Description	Set the gain. Returns True/False if successful.
Usage	**db_basic.db_ lf _rx.set_gain(gain)**
Parameters	***gain***: gain in decibels
Examples	
Note	

1.33.3.8) is_quadrature ()

Type	Sub Function
Description	**Return** True if this daughterboard does quadrature up or down conversion. That is, return True if this board requires both I & Q analog channels. For this board, return False
Usage	**db_basic.db_ lf _rx.is_quadrature()**
Parameters	
Examples	
Note	

1.33.4) db_lf_tx ()

Type	Function
Description	Handler for Low frequency Tx daughterboards
Usage	**db_basic.db_lf_tx(usrp,which)**
Parameters	**usrp**: instance of usrp.sink **which**: which side: 0 or 1 corresponding to TX_A or TX_B respectively
Examples	
Note	Board Technical specifications : Min gain : 0 dB Max gain :20 dB Gain steps : 0.08 dB Min frequency : -32e06 Hz Max frequency : 32e06 Hz Frequency Step : 1e-6 Hz

1.33.4.1) dbid ()

Type	Sub Function
Description	**Return** daughter board ID
Usage	**db_basic.db_ lf _tx.dbid()**
Parameters	
Examples	
Note	

1.33.4.2) name ()

Type	Sub Function
Description	**Return** daughter board name
Usage	**db_basic.db_ lf _tx.name()**
Parameters	
Examples	
Note	

1.33.4.3) side_and_name ()

Type	Sub Function
Description	**Return** daughter board side and name
Usage	**db_basic.db_ lf _tx.side_and_name()**
Parameters	
Examples	
Note	

1.33.4.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this d'board. Returns (min_freq, max_freq, step_size), return type tuple.
Usage	**db_basic.db_ lf _tx.freq_range()**
Parameters	
Examples	
Note	

1.33.4.5) set_freq ()

Type	Sub Function
Description	Set the frequency. **Returns** (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**db_basic.db_ lf _tx.set_freq(target_freq)**
Parameters	**target_freq**: target RF frequency in Hz type freq: float
Examples	
Note	

1.33.4.6) gain_range ()

Type	Sub Function
Description	**Return** range of gain that can be set by this d'board. Returns (min_gain, max_gain, step_size), where gains are expressed in decibels
Usage	**db_basic.db_ lf _tx.gain_range()**
Parameters	
Examples	
Note	

1.33.4.7) set_gain ()

Type	Sub Function
Description	Set the gain. **Returns** True/False if successful.
Usage	**db_basic.db_ lf _tx.set_gain(gain)**
Parameters	***gain***: gain in decibels
Examples	
Note	

1.33.4.8) is_quadrature ()

Type	Sub Function
Description	**Return** True if this daughterboard does quadrature up or down conversion. That is, return True if this board requires both I & Q analog channels. For this board, return True
Usage	**db_basic.db_ lf _tx.is_quadrature()**
Parameters	
Examples	
Note	

1.34) gnuradio/db_dbs_rx.py

Type	Python file
Description	Control DBS receiver based USRP daughterboard.
Note	

1.34.1) db_dbs_rx ()

Type	Function
Description	Control DBS receiver based USRP daughterboard.
Usage	**db_ dbs_rx.db_dbs_rx(usrp,which)**
Parameters	**usrp**: instance of usrp source **which**: which side: 0 or 1 corresponding to side A or side B respectively
Examples	
Note	Board Technical specifications : Min gain : 0 dB Max gain :104 dB Gain steps : 1 dB Min frequency : 500e06 Hz Max frequency : 2600 e06 Hz Frequency Step : 1e6 Hz

1.34.1.1) dbid ()

Type	Sub Function
Description	**Return** daughter board ID
Usage	**db_ dbs_rx.db_ dbs _rx.dbid()**
Parameters	
Examples	
Note	

1.34.1.2) name ()

Type	Sub Function
Description	**Return** daughter board name
Usage	**db_ dbs_rx.db_ dbs _rtx.name()**
Parameters	
Examples	

Note	

1.34.1.3) side_and_name ()

Type	Sub Function
Description	**Return** daughter board side and name
Usage	**db_ dbs_rx.db_ dbs _rx.side_and_name()**
Parameters	
Examples	
Note	

1.34.1.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this d'board. Returns (min_freq, max_freq, step_size), return type tuple.
Usage	**db_ dbs_rx.db_ dbs _rx.freq_range()**
Parameters	
Examples	
Note	

1.34.1.5) set_freq ()

Type	Sub Function
Description	Set the frequency. **Returns** (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**db_ dbs_rx.db_ dbs _rx.set_freq(target_freq)**
Parameters	**taget_freq**: target RF frequency in Hz type freq: float
Examples	
Note	

1.34.1.6) gain_range ()

Type	Sub Function
Description	**Return** range of gain that can be set by this d'board. Returns (min_gain, max_gain, step_size), where gains are expressed in decibels
Usage	**db_ dbs_rx.db_ dbs _rx.gain_range()**
Parameters	
Examples	
Note	

1.34.1.7) set_gain ()

Type	Sub Function
Description	Set the gain. **Returns** True/False if successful
Usage	**db_ dbs_rx.db_ dbs _rx.set_gain(gain)**
Parameters	***gain***: gain in decibels
Examples	
Note	

1.34.1.8) is_quadrature ()

Type	Sub Function
Description	**Return** True if this daughterboard does quadrature up or down conversion. That is, return True if this board requires both I & Q analog channels. For this board, return True.
Usage	**db_ dbs_rx.db_ dbs _rx.is_quadrature()**
Parameters	
Examples	
Note	

1.34.1.9) set_bw ()

Type	Sub Function
Description	Set the bandwidth for the receive channel. Bandwidth should be more than or equal 1MHz and less than 33 MHz
Usage	**db_ dbs_rx.db_ dbs _rx.set_bw(bw)**
Parameters	
Examples	
Note	

1.35) gnuradio/db_flexrf.py
gnuradio/db_flexrf_mimo.py

Type	Python files
Description	Interface functions for all flexrf USRP daughter boards
Note	

1.35.1) db_flexrf_xxxx_tx ()
db_flexrf_xxxx_tx_mimo_x()

Type	Function
Description	Handler for flexrf Tx daughterboards : flex_400_tx flex_900_tx flex_1200_tx flex_1800_tx flex_2400_tx flex_400_tx_mimo_a flex_900_tx_mimo_a flex_1200_tx_mimo_a

	flex_1800_tx_mimo_a flex_2400_tx_mimo_a flex_400_tx_mimo_b flex_900_tx_mimo_b flex_1200_tx_mimo_b flex_1800_tx_mimo_b flex_2400_tx_mimo_b
Usage	**db_flexrf.db_flexrf_xxxx_tx(usrp,which)** or **db_flexrf.db_flexrf_xxxx_tx_mimo_x(usrp,which)**
Parameters	**usrp**: instance of usrp.sink **which**: which side: 0 or 1 corresponding to TX_A or TX_B respectively
Examples	
Note	

Dboard	RF TX power	Min Frequency	Max Frequency	Frequency Step
flex_400_tx	100mW (20 dBm)	400 MHz	500 MHz	1 MHz
flex_900_tx	200 mW (23 dBm)	750 MHz	1050 MHz	4 MHz
flex_1200_tx	200 mW (23 dBm)	1150 MHz	1450 MHz	4 MHz
flex_1800_tx	100 mW (20 dBm)	1500 MHz	2100 MHz	4 MHz
flex_2400_tx	50 mW (17 dBm)	2300 MHz	2900 MHz	4 MHz
flex_400_tx_mimo_a	100mW (20 dBm)	400 MHz	500 MHz	1 MHz
flex_900_tx_mimo_a	200 mW (23 dBm)	750 MHz	1050 MHz	4 MHz
flex_1200_tx_mimo_a	200 mW (23 dBm)	1150 MHz	1450 MHz	4 MHz
flex_1800_tx_mimo_a	100 mW (20 dBm)	1500 MHz	2100 MHz	4 MHz
flex_2400_tx_mimo_a	50 mW (17 dBm)	2300 MHz	2900 MHz	4 MHz
flex_400_tx_mimo_b	100mW (20 dBm)	400 MHz	500 MHz	1 MHz
flex_900_tx_mimo_b	200 mW (23 dBm)	750 MHz	1050 MHz	4 MHz
flex_1200_tx_mimo_b	200 mW (23 dBm)	1150 MHz	1450 MHz	4 MHz
flex_1800_tx_mimo_b	100 mW (20 dBm)	1500 MHz	2100 MHz	4 MHz
flex_2400_tx_mimo_b	50 mW (17 dBm)	2300 MHz	2900 MHz	4 MHz

1.35.1.1) dbid ()

Type	Sub Function
Description	**Return** daughter board ID
Usage	**subdev.dbid()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : **u** : is the USRP sink instance. **subdev_spec**: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.1.2) name ()

Type	Sub Function
Description	**Return** daughter board name
Usage	**subdev.name()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec)

	where : **u** : is the USRP sink instance. **subdev_spec**: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.1.3) side_and_name ()

Type	Sub Function
Description	**Return** daughter board side and name
Usage	**subdev.side_and_name()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.1.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this d'board. Returns (min_freq, max_freq, step_size), return type tuple.
Usage	**subdev.freq_range()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.1.5) set_freq ()

Type	Sub Function
Description	Set the frequency. **Returns** (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**subdev.set_freq(target_freq)**
Parameters	**target_freq**: target RF frequency in Hz type freq: float
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.1.6) is_quadrature ()

Type	Sub Function
Description	**Return** True if this daughterboard does quadrature up or down conversion. That is, **Return** True if this board requires both I & Q analog channels. For this board, return True
Usage	**subdev.is_quadrature()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.1.7) set_auto_tr ()

Type	Sub Function
Description	Enable automatic Transmit/Receive switching (ATR).
Usage	**subdev.set_auto_tr(on)**
Parameters	**on** : bool True or False
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.1.8) set_enable ()

Type	Sub Function
Description	Enable /Disable RF Transmitter
Usage	**subdev.set_enable(on)**
Parameters	**on** : bool True or False
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2) db_flexrf_xxxx_rx ()
db_flexrf_xxxx_rx_mimo_x()

Type	Function
Description	Handler for flexrf Rx daughterboards : flex_400_rx flex_900_rx

	flex_1200_rx flex_1800_rx flex_2400_rx flex_400_rx_mimo_a flex_900_rx_mimo_a flex_1200_rx_mimo_a flex_1800_rx_mimo_a flex_2400_rx_mimo_a flex_400_rx_mimo_b flex_900_rx_mimo_b flex_1200_rx_mimo_b flex_1800_rx_mimo_b flex_2400_rx_mimo_b				
Usage	**db_flexrf.db_flexrf_xxxx_rx(usrp,which)** or **db_flexrf.db_flexrf_xxxx_rx_mimo_x(usrp,which)**				
Parameters	**usrp**: instance of usrp source **which**: which side: 0 or 1 corresponding to RX_A or RX_B respectively				
Examples					
Note					
Dboard	Max Gain	Gain Step	Min Frequency	Max Frequency	Frequency Step
flex_400_rx	65	.035	400 MHz	500 MHz	1 MHz
flex_900_rx	90	.05	750 MHz	1050 MHz	4 MHz
flex_1200_rx	90	.05	1150 MHz	1450 MHz	4 MHz
flex_1800_rx	90	.05	1500 MHz	2100 MHz	4 MHz
flex_2400_rx	90	.05	2300 MHz	2900 MHz	4 MHz
flex_400_rx_mimo_a	65	.035	400 MHz	500 MHz	1 MHz
flex_900_rx_mimo_a	90	.05	750 MHz	1050 MHz	4 MHz
flex_1200_rx_mimo_a	90	.05	1150 MHz	1450 MHz	4 MHz
flex_1800_rx_mimo_a	90	.05	1500 MHz	2100 MHz	4 MHz
flex_2400_rx_mimo_a	90	.05	2300 MHz	2900 MHz	4 MHz
flex_400_rx_mimo_b	65	.035	400 MHz	500 MHz	1 MHz
flex_900_rx_mimo_b	90	.05	750 MHz	1050 MHz	4 MHz
flex_1200_rx_mimo_b	90	.05	1150 MHz	1450 MHz	4 MHz
flex_1800_rx_mimo_b	90	.05	1500 MHz	2100 MHz	4 MHz
flex_2400_rx_mimo_b	90	.05	2300 MHz	2900 MHz	4 MHz

1.35.2.1) dbid ()

Type	Sub Function
Description	**Return** daughter board ID
Usage	**subdev.dbid()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.2) name ()

Type	Sub Function
Description	**Return** daughter board name
Usage	**subdev.name()**
Parameters	

Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.3) side_and_name ()

Type	Sub Function
Description	**Return** daughter board side and name
Usage	**subdev.side_and_name()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this d'board. Returns (min_freq, max_freq, step_size), return type tuple.
Usage	**subdev.freq_range()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.5) set_freq ()

Type	Sub Function
Description	Set the frequency. **Returns** (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**subdev.set_freq(target_freq)**
Parameters	**target_freq**: target RF frequency in Hz type freq: float
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.6) gain_range ()

Type	Sub Function
Description	**Return** range of gain that can be set by this d'board. Returns (min_gain, max_gain, step_size), where gains are expressed in decibels
Usage	**subdev.gain_range()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.7) set_gain ()

Type	Sub Function
Description	Set the gain. **Returns** True/False if successful.
Usage	**subdev.set_gain(gain)**
Parameters	***gain***: gain in decibels
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.8) is_quadrature ()

Type	Sub Function
Description	**Return** True if this daughterboard does quadrature up or down conversion. That is, return True if this board requires both I & Q analog channels. For this board, return True
Usage	**subdev.is_quadrature()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.9) set_auto_tr ()

Type	Sub Function
Description	Enable automatic Transmit/Receive switching (ATR).
Usage	**subdev.set_auto_tr(on)**

Parameters	**on** : bool True or False
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.10) select_rx_antenna ()

Type	Sub Function
Description	Specify which antenna port to use for reception. Choose either 'TX/RX' or 'RX2'
Usage	**subdev.select_rx_antenna(which_antenna)**
Parameters	**which_antenna**: either 'TX/RX' or 'RX2'
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.35.2.11) i_and_q_swapped ()

Type	Sub Function
Description	**Return** True if this is a quadrature device and ADC 0 is Q.
Usage	**subdev.i_and_q_swapped()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.36) gnuradio/db_instantiator.py

Type	Python files
Description	Instantiator for accessing USRP daughter boards
Note	

1.37) gnuradio/db_tv_rx.py

Type	Python file
Description	Control Microtune 4937 based USRP daughterboard
Note	

1.37.1) db_tv_rx ()

Type	Function
Description	Control Microtune 4937 based USRP daughterboard. Three version are invented so far, TV_RX, First IF = 43.75 MHz, second IF = 5.75e6 MHz with second downconversion. TV_RX_REV_2, First IF =44 MHz, second IF = 20 MHz without second downconversion TV_RX_REV_3, First IF = 44 MHz, second IF = 20 MHz without second downconve The The TV_RX 43.75 MHz version has inverted spectrum
Usage	**db_ tv_rx.db_tv_rx(usrp,which,first_IF,second_IF)**
Parameters	**usrp**: instance of usrp source **which**: which side: 0 or 1 corresponding to side A or side B respectively
Examples	
Note	Board Technical specifications : Min gain : 0 dB Max gain :115 dB Gain steps : 1 dB Min frequency : 50e06 Hz Max frequency : 860 e06 Hz Frequency Step : 10e03 Hz

1.37.1.1) dbid ()

Type	Sub Function
Description	**Return** daughter board ID
Usage	**db_ tv_rx.db_ tv _rx.dbid()**
Parameters	
Examples	
Note	

1.37.1.2) name ()

Type	Sub Function
Description	**Return** daughter board name
Usage	**db_ tv _rx.db_ tv _rtx.name()**
Parameters	
Examples	
Note	

1.37.1.3) side_and_name ()

Type	Sub Function
Description	**Return** daughter board side and name
Usage	**db_ tv _rx.db_ tv _rx.side_and_name()**
Parameters	
Examples	
Note	

1.37.1.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this d'board. Returns (min_freq, max_freq, step_size), return type tuple.
Usage	**db_ tv _rx.db_ tv _rx.freq_range()**
Parameters	
Examples	
Note	

1.37.1.5) set_freq ()

Type	Sub Function
Description	Set the frequency. **Returns** (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**db_ tv _rx.db_ tv _rx.set_freq(target_freq)**
Parameters	**targhet_freq**: target RF frequency in Hz type freq: float
Examples	
Note	

1.37.1.6) gain_range ()

Type	Sub Function
Description	**Return** range of gain that can be set by this d'board. Returns (min_gain, max_gain, step_size), where gains are expressed in decibels
Usage	**db_ tv _rx.db_ tv _rx.gain_range()**
Parameters	
Examples	
Note	

1.37.1.7) set_gain ()

Type	Sub Function
Description	Set the gain. **Returns** True/False if successful
Usage	**db_ tv _rx.db_ tv _rx.set_gain(gain)**
Parameters	***gain***: gain in decibels
Examples	
Note	

1.37.1.8) is_quadrature ()

Type	Sub Function
Description	**Return** True if this daughterboard does quadrature up or down conversion. That is, return True if this board requires both I & Q analog channels. For this board, return False
Usage	**db_ tv _rx.db_ tv _rx.is_quadrature()**

Parameters	
Examples	
Note	

1.38) gnuradio/db_wbx.py

Type	Python file
Description	This board is half-duplex. I.e., transmit and receive are mutually exclusive.There is a single LO for both the Tx and Rx sides. The shared control signals are hung off of the Rx side. The shared io controls are duplexed onto the Rx side pins. The wbx_high d'board always needs to be in 'auto_tr_mode'
Note	

1.38.1) db_wbx_lo_rx ()

Type	Function
Description	Handlers for db_wbx_lo_rx dboard
Usage	**db_wbx. db_wbx_lo_rx (usrp,which)**
Parameters	**usrp**: instance of usrp source **which**: which side: 0 or 1 corresponding to side A or side B respectively
Examples	
Note	Board Technical specifications : Min gain : 0 dB Max gain :65 dB Gain steps : .05dB Min frequency : 50e06 Hz Max frequency : 1000 e06 Hz Frequency Step : 16e03 Hz

1.38.1.1) dbid ()

Type	Sub Function
Description	**Return** db_wbx_lo_rx daughter board ID
Usage	**subdev.dbid()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.2) name ()

Type	Sub Function
Description	**Return** db_wbx_lo_rx daughter board name
Usage	**subdev.name()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.3) side_and_name ()

Type	Sub Function
Description	**Return** db_wbx_lo_rx daughter board side and name
Usage	**subdev.side_and_name()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this db_wbx_lo_rx d'board. Returns (min_freq, max_freq, step_size), return type tuple.
Usage	**subdev.freq_range()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.5) set_freq ()

Type	Sub Function
Description	Set the db_wbx_lo_rx frequency. **returns** (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**subdev.set_freq(target_freq)**

Parameters	**target_freq**: target RF frequency in Hz type freq: float
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.6) gain_range ()

Type	Sub Function
Description	**Return** range of gain that can be set by this db_wbx_lo_rx d'board. Returns (min_gain, max_gain, step_size), where gains are expressed in decibels
Usage	**subdev.gain_range()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.7) set_gain ()

Type	Sub Function
Description	Set the gain of db_wbx_lo_rx. **Returns** True/False if successful.
Usage	**subdev.set_gain(gain)**
Parameters	***gain***: gain in decibels
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.8) is_quadrature ()

Type	Sub Function
Description	**Return** True if this db_wbx_lo_rx daughterboard does quadrature up or down conversion. That is, return True if this board requires both I & Q analog channels. For this board, return True
Usage	**subdev.is_quadrature()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.9) set_auto_tr ()

Type	Sub Function
Description	Enable automatic Transmit/Receive switching (ATR).
Usage	**subdev.set_auto_tr(on)**
Parameters	**on** : bool True or False
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.10) select_rx_antenna ()

Type	Sub Function
Description	Specify which antenna port to use for reception. Choose either 'TX/RX' or 'RX2'
Usage	**subdev.select_rx_antenna(which_antenna)**
Parameters	**which_antenna**: either 'TX/RX' or 'RX2'
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.1.11) i_and_q_swapped ()

Type	Sub Function
Description	**Return** True if this is a quadrature device and ADC 0 is Q.
Usage	**subdev.i_and_q_swapped()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP source instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.2) db_wbx_lo_tx ()

Type	Function
Description	Handlers for db_wbx_lo_tx dboard
Usage	**db_wbx. db_wbx_lo_tx (usrp,which)**
Parameters	**usrp**: instance of usrp source

	which: which side: 0 or 1 corresponding to side A or side B respectively
Examples	
Note	Board Technical specifications : Min gain : -56 dB Max gain :0 dB Gain steps : .1dB Min frequency : 50e06 Hz Max frequency : 1000 e06 Hz Frequency Step : 16e03 Hz

1.38.2.1) dbid ()

Type	Sub Function
Description	**Return** db_wbx_lo_tx daughter board ID
Usage	**subdev.dbid()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.2.2) name ()

Type	Sub Function
Description	**Return** db_wbx_lo_tx daughter board name
Usage	**subdev.name()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.2.3) side_and_name ()

Type	Sub Function
Description	**Return** db_wbx_lo_tx daughter board side and name
Usage	**subdev.side_and_name()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.2.4) freq_range ()

Type	Sub Function
Description	**Return** range of frequencies in Hz that can be tuned by this db_wbx_lo_tx d'board. Returns (min_freq, max_freq, step_size), return type tuple.
Usage	**subdev.freq_range()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.2.5) set_freq ()

Type	Sub Function
Description	Set the frequency of db_wbx_lo_tx. **Returns** (ok, actual_baseband_freq) where: ok :bool True or False and indicates success or failure, actual_baseband_freq is the RF frequency that corresponds to DC in the IF.
Usage	**subdev.set_freq(target_freq)**
Parameters	**target_freq**: target RF frequency in Hz type freq: float
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.2.6) is_quadrature ()

Type	Sub Function
Description	**Return** True if this db_wbx_lo_tx daughterboard does quadrature up or down conversion. That is, return True if this board requires both I & Q analog channels. For this board, return True
Usage	**subdev.is_quadrature()**
Parameters	
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.2.7) set_auto_tr ()

Type	Sub Function
Description	Enable automatic Transmit/Receive switching (ATR).
Usage	**subdev.set_auto_tr(on)**
Parameters	**on** : bool True or False
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

1.38.2.8) set_enable ()

Type	Sub Function
Description	Enable /Disable RF Transmitter
Usage	**subdev.set_enable(on)**
Parameters	**on** : bool True or False
Examples	
Note	subdev is the flexrf daughterboard and can be get by : subdev = usrp.selected_subdev(u, subdev_spec) where : u : is the USRP sink instance. subdev_spec: is the tuple (side, subdev), where side is 0 (Side A) or 1 (Side B) and subdev is 0 (Input I) or 1 (input Q)

2) usrpm package

Description	

2.1) usrpm/usrp_dbid.py

Type	Python file
Description	This file contains all USRP daughter boards ID's invented yet. These are : BASIC_TX = 0x0000 BASIC_RX = 0x0001 DBS_RX = 0x0002 TV_RX = 0x0003 FLEX_400_RX = 0x0004 FLEX_900_RX = 0x0005 FLEX_1200_RX = 0x0006 FLEX_2400_RX = 0x0007 FLEX_400_TX = 0x0008 FLEX_900_TX = 0x0009 FLEX_1200_TX = 0x000a FLEX_2400_TX = 0x000b TV_RX_REV_2 = 0x000c DBS_RX_REV_2_1 = 0x000d

	LF_TX = 0x000e LF_RX = 0x000f FLEX_400_RX_MIMO_A = 0x0014 FLEX_900_RX_MIMO_A = 0x0015 FLEX_1200_RX_MIMO_A = 0x0016 FLEX_2400_RX_MIMO_A = 0x0017 FLEX_400_TX_MIMO_A = 0x0018 FLEX_900_TX_MIMO_A = 0x0019 FLEX_1200_TX_MIMO_A = 0x001a FLEX_2400_TX_MIMO_A = 0x001b FLEX_400_RX_MIMO_B = 0x0024 FLEX_900_RX_MIMO_B = 0x0025 FLEX_1200_RX_MIMO_B = 0x0026 FLEX_2400_RX_MIMO_B = 0x0027 FLEX_400_TX_MIMO_B = 0x0028 FLEX_900_TX_MIMO_B = 0x0029 FLEX_1200_TX_MIMO_B = 0x002a FLEX_2400_TX_MIMO_B = 0x002b FLEX_1800_RX = 0x0030 FLEX_1800_TX = 0x0031 FLEX_1800_RX_MIMO_A = 0x0032 FLEX_1800_TX_MIMO_A = 0x0033 FLEX_1800_RX_MIMO_B = 0x0034 FLEX_1800_TX_MIMO_B = 0x0035 TV_RX_REV_3 = 0x0040 WBX_LO_TX = 0x0050 WBX_LO_RX = 0x0051 EXPERIMENTAL_TX = 0xfffe EXPERIMENTAL_RX = 0xffff
Note	

2.2) usrpm/usrp_prims.py

Type	Python file
Description	This file was automatically generated by SWIG
Note	

2.3) usrpm/usrp_fpga_regs.py

Type	Python file
Description	This file contains all USRP fpga registers. These are : FR_TX_SAMPLE_RATE_DIV FR_RX_SAMPLE_RATE_DIV FR_MASTER_CTRL bmFR_MC_ENABLE_TX bmFR_MC_ENABLE_RX bmFR_MC_RESET_TX bmFR_MC_RESET_RX FR_OE_0 FR_OE_1 FR_OE_2 FR_OE_3 FR_IO_0 FR_IO_1 FR_IO_2 FR_IO_3

	FR_MODE bmFR_MODE_NORMAL bmFR_MODE_LOOPBACK bmFR_MODE_RX_COUNTING bmFR_MODE_RX_COUNTING_32BIT FR_DEBUG_EN bmFR_DEBUG_EN_TX_A bmFR_DEBUG_EN_RX_A bmFR_DEBUG_EN_TX_B bmFR_DEBUG_EN_RX_B FR_DC_OFFSET_CL_EN FR_ADC_OFFSET_0 FR_ADC_OFFSET_1 FR_ADC_OFFSET_2 FR_ADC_OFFSET_3 FR_ATR_MASK_0 FR_ATR_TXVAL_0 FR_ATR_RXVAL_0 FR_ATR_MASK_1 FR_ATR_TXVAL_1 FR_ATR_RXVAL_1 FR_ATR_MASK_2 FR_ATR_TXVAL_2 FR_ATR_RXVAL_2 FR_ATR_MASK_3 FR_ATR_TXVAL_3 FR_ATR_RXVAL_3 FR_ATR_TX_DELAY FR_ATR_RX_DELAY FR_INTERP_RATE FR_DECIM_RATE FR_RX_FREQ_0 FR_RX_FREQ_1 FR_RX_FREQ_2 FR_RX_FREQ_3 FR_RX_MUX FR_TX_MUX FR_TX_A_REFCLK FR_RX_A_REFCLK FR_TX_B_REFCLK FR_RX_B_REFCLK bmFR_REFCLK_EN bmFR_REFCLK_DIVISOR_MASK FR_RX_PHASE_0 FR_RX_PHASE_1 FR_RX_PHASE_2 FR_RX_PHASE_3 FR_TX_FORMAT bmFR_TX_FORMAT_16_IQ FR_RX_FORMAT bmFR_RX_FORMAT_SHIFT_MASK bmFR_RX_FORMAT_SHIFT_SHIFT bmFR_RX_FORMAT_WIDTH_MASK bmFR_RX_FORMAT_WIDTH_SHIFT bmFR_RX_FORMAT_WANT_Q bmFR_RX_FORMAT_BYPASS_HB FR_USER_0 FR_USER_1 FR_USER_2 FR_USER_3 FR_USER_4 FR_USER_5 FR_USER_6 FR_USER_7

	FR_USER_8 FR_USER_9 FR_USER_10 FR_USER_11 FR_USER_12 FR_USER_13 FR_USER_14 FR_USER_15 FR_USER_16 FR_USER_17 FR_USER_18 FR_USER_19 FR_USER_20 FR_USER_21 FR_USER_22 FR_USER_23 FR_USER_24 FR_USER_25 FR_USER_26 FR_USER_27 FR_USER_28 FR_USER_29 FR_USER_30 FR_USER_31 FR_RX_MASTER_SLAVE bmFR_RX_SYNC bmFR_RX_SYNC_MASTER bmFR_RX_SYNC_SLAVE bmFR_RX_SYNC_INPUT_IOPIN bmFR_RX_SYNC_OUTPUT_IOPIN FR_RB_IO_RX_A_IO_TX_A FR_RB_IO_RX_B_IO_TX_B FR_RB_CAPS bmFR_RB_CAPS_NDDC_MASK bmFR_RB_CAPS_NDDC_SHIFT bmFR_RB_CAPS_RX_HAS_HALFBAND bmFR_RB_CAPS_NDUC_MASK bmFR_RB_CAPS_NDUC_SHIFT bmFR_RB_CAPS_TX_HAS_HALFBAND
Note	

Index

A

B

C

D

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W