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

gnuradio.digital

class `gnuradio.digital.`**`constellation`**(**args*, ***kwargs*)

An abstracted constellation object.

The constellation objects hold the necessary information to pass around constellation information for modulators and demodulators. These objects contain the mapping between the bits and the constellation points used to represent them as well as methods for slicing the symbol space. Various implementations are possible for efficiency and ease of use.

Standard constellations (BPSK, QPSK, QAM, etc) can be inherited from this class and overloaded to perform optimized slicing and constellation mappings.

class `gnuradio.digital.`**`lfsr`**(*mask*, *seed*, *reg_len*)

Fibonacci Linear Feedback Shift Register using specified polynomial mask.

Generates a maximal length pseudo-random sequence of length 2^degree-1

Constructor: digital::lfsr(int mask, int seed, int reg_len);

Some common masks might be: x^4 + x^3 + x^0 = 0x19 x^5 + x^3 + x^0 = 0x29 x^6 + x^5 + x^0 = 0x61

see for more explanation.

next_bit() - Standard LFSR operation

next_bit_scramble(unsigned char input) - Scramble an input stream

next_bit_descramble(unsigned char input) - Descramble an input stream

See for operation of these last two functions (see multiplicative scrambler.)

Parameters:
- **mask** –
  - polynomial coefficients representing the locations of feedback taps from a shift register which are xor'ed together to form the new high order bit.
- **seed** –
  - the initialization vector placed into the register during initialization. Low order bit corresponds to x^0 coefficient the first to be shifted as output.
- **reg_len** –
  - specifies the length of the feedback shift register to be used. During each iteration, the register is rightshifted one and the new bit is placed in bit reg_len. reg_len should generally be at least order(mask) + 1

class `gnuradio.digital.`**`mpsk_snr_est`**(*alpha*)

A parent class for SNR estimators, specifically for M-PSK signals in AWGN channels.

`gnuradio.digital.`**`simple_framer`**

alias of `make`

class `gnuradio.digital.`**`bpsk_mod`**(*mod_code=None*, *differential=False*, **args*, ***kwargs*)

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.

Parameters:
- **mod_code** – Argument is not used. It exists purely to simplify generation of the block in grc.
- **differential** – Whether to use differential encoding (boolean).
- **samples_per_symbol** – samples per baud >= 2 (float)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

class `gnuradio.digital.`**`bpsk_demod`**(*mod_code=None*, *differential=False*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered BPSK demodulation.

The input is the complex modulated signal at baseband and the output is a stream of bits packed 1 bit per byte (LSB)

Parameters:

- **mod_code** – Argument is not used. It exists purely to simplify generation of the block in grc.
- **differential** – whether to use differential encoding (boolean)
- **samples_per_symbol** – samples per baud >= 2 (float)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **freq_bw** – loop filter lock-in bandwidth (float)
- **timing_bw** – timing recovery loop lock-in bandwidth (float)
- **phase_bw** – phase recovery loop bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

class `gnuradio.digital.`**`dbpsk_mod`**(*mod_code=None*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered DBPSK modulation.

The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.

Parameters:

- **mod_code** – Argument is not used. It exists purely to simplify generation of the block in grc.
- **samples_per_symbol** – samples per baud >= 2 (float)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

class `gnuradio.digital.`**`dbpsk_demod`**(*mod_code=None*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered DBPSK demodulation.

The input is the complex modulated signal at baseband and the output is a stream of bits packed 1 bit per byte (LSB)

Parameters:

- **mod_code** – Argument is not used. It exists purely to simplify generation of the block in grc.
- **samples_per_symbol** – samples per baud >= 2 (float)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **freq_bw** – loop filter lock-in bandwidth (float)
- **timing_bw** – timing recovery loop lock-in bandwidth (float)
- **phase_bw** – phase recovery loop bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

`gnuradio.digital.`**`crc32`**(*unsigned char const * buf*, *size_t len*) → unsigned int

crc32(std::string const buf) -> unsigned int

`gnuradio.digital.`**`update_crc32`**(*unsigned int crc*, *unsigned char const * buf*, *size_t len*) → unsigned int

update_crc32(unsigned int crc, std::string const buf) -> unsigned int

update running CRC-32

Update a running CRC with the bytes buf[0..len-1] The CRC should be initialized to all 1's, and the transmitted value is the 1's complement of the final running CRC. The resulting CRC should be transmitted in big endian order.

`gnuradio.digital.`**`constellation_map_generator`**(*basis_cpoints*, *basis_symbols*, *k*, *pi*)

Uses the a basis constellation provided (e.g., from psk_constellation.psk_4()) and the the k and permutation index (pi) to generate a new Gray-coded symbol map to the constellation points provided in the basis.

The basis_cpoints are the constellation points of the basis constellation, and basis_symbols are the symbols that correspond to the constellation points.

The selection of k and pi will provide an automorphism the hyperoctahedral group of the basis constellation.

This function returns a tuple of (constellation_points, symbol_map). The constellation_points is a list of the constellation points in complex space and the symbol_map is a list of the log2(M)-bit symbols for the constellation points (i.e., symbol_map[i] are the bits associated with constellation_points[i]).

class `gnuradio.digital.`**`cpm_mod`**(*samples_per_symbol=2*, *bits_per_symbol=1*, *h_numerator=1*, *h_denominator=2*, *cpm_type=0*, *bt=0.35*, *symbols_per_pulse=1*, *generic_taps=array([9.22337204e+18])*, *verbose=False*, *log=False*)

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)

Parameters:

- **samples_per_symbol** – samples per baud >= 2 (integer)
- **bits_per_symbol** – bits per symbol (integer)
- **h_numerator** – numerator of modulation index (integer)
- **h_denominator** – denominator of modulation index (numerator and denominator must be relative primes) (integer)
- **cpm_type** – supported types are: 0=CPFSK, 1=GMSK, 2=RC, 3=GENERAL (integer)
- **bt** – bandwidth symbol time product for GMSK (float)
- **symbols_per_pulse** – shaping pulse duration in symbols (integer)
- **generic_taps** – define a generic CPM pulse shape (sum = samples_per_symbol/2) (list/array of floats)
- **verbose** – Print information about modulator? (boolean)
- **debug** – Print modulation data to files? (boolean)

`gnuradio.digital.`**`gen_and_append_crc32`**(*s*)

`gnuradio.digital.`**`check_crc32`**(*s*)

class `gnuradio.digital.`**`generic_mod`**(*constellation*, *differential=False*, *samples_per_symbol=2*, *pre_diff_code=True*, *excess_bw=0.35*, *verbose=False*, *log=False*)

Hierarchical block for RRC-filtered differential generic modulation.

The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.

Parameters:

- **constellation** – determines the modulation type (gnuradio.digital.digital_constellation)
- **samples_per_symbol** – samples per baud >= 2 (float)
- **differential** – whether to use differential encoding (boolean)
- **pre_diff_code** – whether to use apply a pre-differential mapping (boolean)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

class `gnuradio.digital.`**`generic_demod`**(*constellation*, *differential=False*, *samples_per_symbol=2*, *pre_diff_code=True*, *excess_bw=0.35*, *freq_bw=0.06283185307179587*, *timing_bw=0.06283185307179587*, *phase_bw=0.06283185307179587*, *verbose=False*, *log=False*)

Hierarchical block for RRC-filtered differential generic demodulation.

The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (LSB)

Parameters:

- **constellation** – determines the modulation type (gnuradio.digital.digital_constellation)
- **samples_per_symbol** – samples per baud >= 2 (float)
- **differential** – whether to use differential encoding (boolean)
- **pre_diff_code** – whether to use apply a pre-differential mapping (boolean)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **freq_bw** – loop filter lock-in bandwidth (float)
- **timing_bw** – timing recovery loop lock-in bandwidth (float)
- **phase_bw** – phase recovery loop bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

class `gnuradio.digital.`**`gfsk_mod`**(*samples_per_symbol=2*, *sensitivity=1*, *bt=0.35*, *verbose=False*, *log=False*)

class `gnuradio.digital.`**`gfsk_demod`**(*samples_per_symbol=2*, *sensitivity=1*, *gain_mu=None*, *mu=0.5*, *omega_relative_limit=0.005*, *freq_error=0.0*, *verbose=False*, *log=False*)

class `gnuradio.digital.`**`gmsk_mod`**(*samples_per_symbol=2*, *bt=0.35*, *verbose=False*, *log=False*)

Hierarchical block for Gaussian Minimum Shift Key (GMSK) modulation.

The input is a byte stream (unsigned char with packed bits) and the output is the complex modulated signal at baseband.

Parameters:

- **samples_per_symbol** – samples per baud >= 2 (integer)
- **bt** – Gaussian filter bandwidth * symbol time (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Print modulation data to files? (boolean)

class `gnuradio.digital.`**`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*)

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)

Parameters:

- **samples_per_symbol** – samples per baud (integer)
- **gain_mu** – controls rate of mu adjustment (float)
- **mu** – fractional delay [0.0, 1.0] (float)
- **omega_relative_limit** – sets max variation in omega (float)
- **freq_error** – bit rate error as a fraction (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Print modualtion data to files? (boolean)

`gnuradio.digital.`**`type_1_mods`**()

`gnuradio.digital.`**`add_type_1_mod`**(*name*, *mod_class*)

`gnuradio.digital.`**`type_1_demods`**()

`gnuradio.digital.`**`add_type_1_demod`**(*name*, *demod_class*)

`gnuradio.digital.`**`type_1_constellations`**()

`gnuradio.digital.`**`add_type_1_constellation`**(*name*, *constellation*)

`gnuradio.digital.`**`extract_kwargs_from_options`**(*function*, *excluded_args*, *options*)

Given a function, a list of excluded arguments and the result of parsing command line options, create a dictionary of key word arguments suitable for passing to the function. The dictionary will be populated with key/value pairs where the keys are those that are common to the function's argument list (minus the excluded_args) and the attributes in options. The values are the corresponding values from options unless that value is None. In that case, the corresponding dictionary entry is not populated.

(This allows different modulations that have the same parameter names, but different default values to coexist. The downside is that –help in the option parser will list the default as None, but in that case the default provided in the __init__ argument list will be used since there is no kwargs entry.)

Parameters:
- **function** – the function whose parameter list will be examined
- **excluded_args** – function arguments that are NOT to be added to the dictionary (sequence of strings)
- **options** – result of command argument parsing (optparse.Values)

`gnuradio.digital.`**`extract_kwargs_from_options_for_class`**(*cls*, *options*)

Given command line options, create dictionary suitable for passing to __init__

`gnuradio.digital.ofdm_packet_utils.`**`conv_packed_binary_string_to_1_0_string`**(*s*)

'¯' –> '10101111'

`gnuradio.digital.ofdm_packet_utils.`**`conv_1_0_string_to_packed_binary_string`**(*s*)

'10101111' -> ('¯', 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.

`gnuradio.digital.ofdm_packet_utils.`**`is_1_0_string`**(*s*)

`gnuradio.digital.ofdm_packet_utils.`**`string_to_hex_list`**(*s*)

`gnuradio.digital.ofdm_packet_utils.`**`whiten`**(*s*, *o*)

`gnuradio.digital.ofdm_packet_utils.`**`dewhiten`**(*s*, *o*)

`gnuradio.digital.ofdm_packet_utils.`**`make_header`**(*payload_len*, *whitener_offset=0*)

`gnuradio.digital.ofdm_packet_utils.`**`make_packet`**(*payload*, *samples_per_symbol*, *bits_per_symbol*, *pad_for_usrp=True*, *whitener_offset=0*, *whitening=True*)

Build a packet, given access code, payload, and whitener offset

Parameters:
- **payload** – packet payload, len [0, 4096]
- **samples_per_symbol** – samples per symbol (needed for padding calculation) (int)
- **bits_per_symbol** – (needed for padding calculation) (int)
- **whitener_offset** – offset into whitener string to use [0-16)
- **whitening** – Turn whitener on or off (bool)

Packet will have access code at the beginning, followed by length, payload and finally CRC-32.

`gnuradio.digital.ofdm_packet_utils.`**`unmake_packet`**(*whitened_payload_with_crc*, *whitener_offset=0*, *dewhitening=1*)

Return (ok, payload)

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

class `gnuradio.digital.`**`ofdm_mod`**(*options*, *msgq_limit=2*, *pad_for_usrp=True*)

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

class `gnuradio.digital.`**`ofdm_demod`**(*options*, *callback=None*)

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.

class `gnuradio.digital.`**`ofdm_receiver`**(*fft_length*, *cp_length*, *occupied_tones*, *snr*, *ks*, *logging=False*)

Performs receiver synchronization on OFDM symbols.

The receiver performs channel filtering as well as symbol, frequency, and phase synchronization. The synchronization routines are available in three flavors: preamble correlator (Schmidl and Cox), modifid preamble correlator with autocorrelation (not yet working), and cyclic prefix correlator (Van de Beeks).

class `gnuradio.digital.`**`ofdm_sync_fixed`**(*fft_length*, *cp_length*, *nsymbols*, *freq_offset*, *logging=False*)

class `gnuradio.digital.`**`ofdm_sync_ml`**(*fft_length*, *cp_length*, *snr*, *kstime*, *logging*)

class `gnuradio.digital.`**`ofdm_sync_pnac`**(*fft_length*, *cp_length*, *kstime*, *logging=False*)

class `gnuradio.digital.`**`ofdm_sync_pn`**(*fft_length*, *cp_length*, *logging=False*)

class `gnuradio.digital.`**`ofdm_tx`**(*fft_len=64, cp_len=16, packet_length_tag_key='packet_length', occupied_carriers=([-26, -25, -24, -23, -22, -20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -6, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26],), pilot_carriers=((-21, -7, 7, 21),), pilot_symbols=((1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1)), bps_header=1, bps_payload=1, sync_word1=None, sync_word2=None, rolloff=0, debug_log=False, scramble_bits=False*)

Hierarchical block for OFDM modulation.

The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.

class `gnuradio.digital.`**`ofdm_rx`**(*fft_len=64, cp_len=16, frame_length_tag_key='frame_length', packet_length_tag_key='packet_length', packet_num_tag_key='packet_num', occupied_carriers=([-26, -25, -24, -23, -22, -20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -6, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26],), pilot_carriers=((-21, -7, 7, 21),), pilot_symbols=((1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (1, 1, 1, -1), (1, 1, 1, -1), (1, 1, 1, -1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1), (-1, -1, -1, 1)), bps_header=1,* *bps_payload=1, sync_word1=None, sync_word2=None, debug_log=False, scramble_bits=False*)

Hierarchical block for OFDM demodulation.

The input is a complex baseband signal (e.g. from a UHD source). The detected packets are output as a stream of packed bits on the output.

`gnuradio.digital.packet_utils.`**`conv_packed_binary_string_to_1_0_string`**(*s*)

'¯' –> '10101111'

`gnuradio.digital.packet_utils.`**`conv_1_0_string_to_packed_binary_string`**(*s*)

'10101111' -> ('¯', 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.

`gnuradio.digital.packet_utils.`**`is_1_0_string`**(*s*)

`gnuradio.digital.packet_utils.`**`string_to_hex_list`**(*s*)

`gnuradio.digital.packet_utils.`**`whiten`**(*s*, *o*)

`gnuradio.digital.packet_utils.`**`dewhiten`**(*s*, *o*)

`gnuradio.digital.packet_utils.`**`make_header`**(*payload_len*, *whitener_offset=0*)

`gnuradio.digital.packet_utils.`**`make_packet`**(*payload*, *samples_per_symbol*, *bits_per_symbol*, *preamble='1010010011110010'*, *access_code='1010110011011101101001001110001011110010100011000010000011111100'*, *pad_for_usrp=True*, *whitener_offset=0*, *whitening=True*, *calc_crc=True*)

Build a packet, given access code, payload, and whitener offset

Parameters:	
	• **payload** – packet payload, len [0, 4096]
	• **samples_per_symbol** – samples per symbol (needed for padding calculation) (int)
	• **bits_per_symbol** – (needed for padding calculation) (int)
	• **preamble** – string of ascii 0's and 1's
	• **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(512 bytes)
	• **whitener_offset** – offset into whitener string to use [0-16)
	• **whitening** – Whether to turn on data whitening(scrambling) (boolean)
	• **calc_crc** – Whether to calculate CRC32 or not (boolean)

Packet will have access code at the beginning, followed by length, payload and finally CRC-32.

`gnuradio.digital.packet_utils.`**`unmake_packet`**(*whitened_payload_with_crc*, *whitener_offset=0*, *dewhitening=True*, *check_crc=True*)

Return (ok, payload)

Parameters:	
	• **whitened_payload_with_crc** – string
	• **whitener_offset** – integer offset into whitener table
	• **dewhitening** – True if we should run this through the dewhitener
	• **check_crc** – True if we should check the CRC of the packet

class `gnuradio.digital.`**`mod_pkts`**(*modulator*, *preamble=None*, *access_code=None*, *msgq_limit=2*, *pad_for_usrp=True*, *use_whitener_offset=False*, *modulate=True*)

Wrap an arbitrary digital modulator in our packet handling framework.

Send packets by calling send_pkt

class `gnuradio.digital.`**`demod_pkts`**(*demodulator*, *access_code=None*, *callback=None*, *threshold=-1*)

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.

gnuradio.digital.**psk_2_0x0**()

0 | 1

gnuradio.digital.**psk_2_0x1**()

1 | 0

gnuradio.digital.**sd_psk_2_0x0**(*x*, *Es=1*)

0 | 1

gnuradio.digital.**sd_psk_2_0x1**(*x*, *Es=1*)

1 | 0

gnuradio.digital.**psk_4_0x0_0_1**()

10 | 11
———
00 | 01

gnuradio.digital.**psk_4_0x1_0_1**()

11 | 10
———
01 | 00

gnuradio.digital.**psk_4_0x2_0_1**()

00 | 01
———
10 | 11

gnuradio.digital.**psk_4_0x3_0_1**()

01 | 00
———
11 | 10

gnuradio.digital.**psk_4_0x0_1_0**()

01 | 11
———
00 | 10

gnuradio.digital.**psk_4_0x1_1_0**()

00 | 10
———
01 | 11

gnuradio.digital.**psk_4_0x2_1_0**()

11 | 01
———
10 | 00

gnuradio.digital.**psk_4_0x3_1_0**()

10 | 00
———
11 | 01

gnuradio.digital.**sd_psk_4_0x0_0_1**(*x*, *Es=1*)

10 | 11
———
00 | 01

gnuradio.digital.**sd_psk_4_0x1_0_1**(*x*, *Es=1*)

11 | 10
———

01 | 00

gnuradio.digital.**sd_psk_4_0x2_0_1**(*x*, *Es=1*)

00 | 01

—–

10 | 11

gnuradio.digital.**sd_psk_4_0x3_0_1**(*x*, *Es=1*)

01 | 00

—–

11 | 10

gnuradio.digital.**sd_psk_4_0x0_1_0**(*x*, *Es=1*)

01 | 11

—–

00 | 10

gnuradio.digital.**sd_psk_4_0x1_1_0**(*x*, *Es=1*)

00 | 10

—–

01 | 11

gnuradio.digital.**sd_psk_4_0x2_1_0**(*x*, *Es=1*)

11 | 01

—–

10 | 00

gnuradio.digital.**sd_psk_4_0x3_1_0**(*x*, *Es=1*)

10 | 00

—–

11 | 01

gnuradio.digital.**psk_constellation**(*m=4*, *mod_code='gray'*, *differential=True*)

Creates a PSK constellation object.

class gnuradio.digital.**psk_mod**(*constellation_points=4*, *mod_code='gray'*, *differential=True*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered PSK modulation.

The input is a byte stream (unsigned char), treated as a series of packed symbols. Symbols are grouped from MSB to LSB.

The output is the complex modulated signal at baseband, with a given number of samples per symbol.

If “Samples/Symbol” is 2, and “Number of Constellation Points” is 4, a single byte contains four symbols, and will produce eight samples.

Parameters:
- **constellation_points** – Number of constellation points (must be a power of two) (integer).
- **mod_code** – Whether to use a gray_code (digital.mod_codes.GRAY_CODE) or not (digital.mod_codes.NO_CODE).
- **differential** – Whether to use differential encoding (boolean).
- **samples_per_symbol** – samples per baud >= 2 (float)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

class gnuradio.digital.**psk_demod**(*constellation_points=4*, *mod_code='gray'*, *differential=True*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered PSK modulation.

The input is a complex modulated signal at baseband.

The output is a stream of bytes, each representing a recovered bit. The most significant bit is reported first.

Parameters:

- **constellation_points** – Number of constellation points (must be a power of two) (integer).
- **mod_code** – Whether to use a gray_code (digital.mod_codes.GRAY_CODE) or not (digital.mod_codes.NO_CODE).
- **differential** – Whether to use differential encoding (boolean).
- **samples_per_symbol** – samples per baud >= 2 (float)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

gnuradio.digital.**qam_16_0x0_0_1_2_3**()

0010 0110 | 1110 1010

0011 0111 | 1111 1011

0001 0101 | 1101 1001

0000 0100 | 1100 1000

gnuradio.digital.**qam_16_0x1_0_1_2_3**()

0011 0111 | 1111 1011

0010 0110 | 1110 1010

0000 0100 | 1100 1000

0001 0101 | 1101 1001

gnuradio.digital.**qam_16_0x2_0_1_2_3**()

0000 0100 | 1100 1000

0001 0101 | 1101 1001

0011 0111 | 1111 1011

0010 0110 | 1110 1010

gnuradio.digital.**qam_16_0x3_0_1_2_3**()

0001 0101 | 1101 1001

0000 0100 | 1100 1000

0010 0110 | 1110 1010

0011 0111 | 1111 1011

gnuradio.digital.**qam_16_0x0_1_0_2_3**()

0001 0101 | 1101 1001

0011 0111 | 1111 1011

0010 0110 | 1110 1010

0000 0100 | 1100 1000

gnuradio.digital.**qam_16_0x1_1_0_2_3**()

0000 0100 | 1100 1000

0010 0110 | 1110 1010

0011 0111 | 1111 1011

0001 0101 | 1101 1001

gnuradio.digital.**qam_16_0x2_1_0_2_3**()

0011 0111 | 1111 1011

0001 0101 | 1101 1001

0000 0100 | 1100 1000

0010 0110 | 1110 1010

gnuradio.digital.**qam_16_0x3_1_0_2_3**()

0010 0110 | 1110 1010

0000 0100 | 1100 1000

0001 0101 | 1101 1001

0011 0111 | 1111 1011

gnuradio.digital.**sd_qam_16_0x0_0_1_2_3**(*x*, *Es=1*)

Soft bit LUT generator for constellation:

0010 0110 | 1110 1010

0011 0111 | 1111 1011

0001 0101 | 1101 1001

0000 0100 | 1100 1000

gnuradio.digital.**sd_qam_16_0x1_0_1_2_3**(*x*, *Es=1*)

Soft bit LUT generator for constellation:

0011 0111 | 1111 1011

0010 0110 | 1110 1010

0000 0100 | 1100 1000

0001 0101 | 1101 1001

gnuradio.digital.**sd_qam_16_0x2_0_1_2_3**(*x*, *Es=1*)

Soft bit LUT generator for constellation:

0000 0100 | 1100 1000

0001 0101 | 1101 1001

0011 0111 | 1111 1011

0010 0110 | 1110 1010

gnuradio.digital.**sd_qam_16_0x3_0_1_2_3**(*x*, *Es=1*)

Soft bit LUT generator for constellation:

0001 0101 | 1101 1001

0000 0100 | 1100 1000

0010 0110 | 1110 1010

0011 0111 | 1111 1011

gnuradio.digital.**sd_qam_16_0x0_1_0_2_3**(*x*, *Es=1*)

Soft bit LUT generator for constellation:

0001 0101 | 1101 1001

0011 0111 | 1111 1011

0010 0110 | 1110 1010

0000 0100 | 1100 1000

gnuradio.digital.**sd_qam_16_0x1_1_0_2_3**(*x*, *Es=1*)

Soft bit LUT generator for constellation:

0000 0100 | 1100 1000

0010 0110 | 1110 1010

0011 0111 | 1111 1011

0001 0101 | 1101 1001

gnuradio.digital.**sd_qam_16_0x2_1_0_2_3**(*x*, *Es=1*)

Soft bit LUT generator for constellation:

0011 0111 | 1111 1011

0001 0101 | 1101 1001

0000 0100 | 1100 1000

0010 0110 | 1110 1010

gnuradio.digital.**sd_qam_16_0x3_1_0_2_3**(*x*, *Es=1*)

Soft bit LUT generator for constellation:

0010 0110 | 1110 1010

0000 0100 | 1100 1000

0001 0101 | 1101 1001

0011 0111 | 1111 1011

gnuradio.digital.**qam32_holeinside_constellation**(*large_ampls_to_corners=False*)

gnuradio.digital.**make_differential_constellation**(*m*, *gray_coded*)

Create a constellation with m possible symbols where m must be a power of 4.

Points are laid out in a square grid.

Bits referring to the quadrant are differentilly encoded, remaining bits are gray coded.

gnuradio.digital.**make_non_differential_constellation**(*m*, *gray_coded*)

gnuradio.digital.**qam_constellation**(*constellation_points=16*, *differential=True*, *mod_code='none'*, *large_ampls_to_corners=False*)

Creates a QAM constellation object.

If large_ampls_to_corners=True then sectors that are probably occupied due to a phase offset, are not mapped to the closest constellation point. Rather we take into account the fact that a phase offset is probably the problem and map them to the closest corner point. It’s a bit hackish but it seems to improve frequency locking.

class gnuradio.digital.**qam_mod**(*constellation_points=16*, *differential=True*, *mod_code='none'*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered QAM modulation.

The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.

Parameters:

- **constellation_points** – Number of constellation points (must be a power of four) (integer).
- **mod_code** – Whether to use a gray_code (digital.mod_codes.GRAY_CODE) or not (digital.mod_codes.NO_CODE).
- **differential** – Whether to use differential encoding (boolean).
- **samples_per_symbol** – samples per baud >= 2 (float)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

class gnuradio.digital.**qam_demod**(*constellation_points=16*, *differential=True*, *mod_code='none'*, *large_ampls_to_corner=False*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered QAM modulation.

The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.

Parameters:

- **constellation_points** – Number of constellation points (must be a power of four) (integer).
- **mod_code** – Whether to use a gray_code (digital.mod_codes.GRAY_CODE) or not (digital.mod_codes.NO_CODE).
- **differential** – Whether to use differential encoding (boolean).
- **samples_per_symbol** – samples per baud >= 2 (float)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

gnuradio.digital.**qpsk_constellation**(*mod_code='gray'*)

Creates a QPSK constellation.

class gnuradio.digital.**qpsk_mod**(*mod_code='gray'*, *differential=False*, **args*, ***kwargs*)

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.

Parameters:

- **mod_code** – Whether to use a gray_code (digital.mod_codes.GRAY_CODE) or not (digital.mod_codes.NO_CODE).
- **differential** – Whether to use differential encoding (boolean).
- **samples_per_symbol** – samples per baud >= 2 (float)
- **excess_bw** – Root-raised cosine filter excess bandwidth (float)
- **verbose** – Print information about modulator? (boolean)
- **log** – Log modulation data to files? (boolean)

class gnuradio.digital.**qpsk_demod**(*mod_code='gray'*, *differential=False*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered QPSK demodulation.

The input is the complex modulated signal at baseband and the output is a stream of bits packed 1 bit per byte (LSB)

Parameters:	
	• **mod_code** – Whether to use a gray_code (digital.mod_codes.GRAY_CODE) or not (digital.mod_codes.NO_CODE). • **differential** – Whether to use differential encoding (boolean). • **samples_per_symbol** – samples per baud >= 2 (float) • **excess_bw** – Root-raised cosine filter excess bandwidth (float) • **freq_bw** – loop filter lock-in bandwidth (float) • **timing_bw** – timing recovery loop lock-in bandwidth (float) • **phase_bw** – phase recovery loop bandwidth (float) • **verbose** – Print information about modulator? (boolean) • **log** – Log modulation data to files? (boolean)

`gnuradio.digital.`**`dqpsk_constellation`**(*mod_code='gray'*)

class `gnuradio.digital.`**`dqpsk_mod`**(*mod_code='gray'*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered DQPSK modulation.

The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.

Parameters:	
	• **mod_code** – Whether to use a gray_code (digital.mod_codes.GRAY_CODE) or not (digital.mod_codes.NO_CODE). • **samples_per_symbol** – samples per baud >= 2 (float) • **excess_bw** – Root-raised cosine filter excess bandwidth (float) • **verbose** – Print information about modulator? (boolean) • **log** – Log modulation data to files? (boolean)

class `gnuradio.digital.`**`dqpsk_demod`**(*mod_code='gray'*, **args*, ***kwargs*)

Hierarchical block for RRC-filtered DQPSK demodulation.

The input is the complex modulated signal at baseband and the output is a stream of bits packed 1 bit per byte (LSB)

Parameters:	
	• **mod_code** – Whether to use a gray_code (digital.mod_codes.GRAY_CODE) or not (digital.mod_codes.NO_CODE). • **samples_per_symbol** – samples per baud >= 2 (float) • **excess_bw** – Root-raised cosine filter excess bandwidth (float) • **freq_bw** – loop filter lock-in bandwidth (float) • **timing_bw** – timing recovery loop lock-in bandwidth (float) • **phase_bw** – phase recovery loop bandwidth (float) • **verbose** – Print information about modulator? (boolean) • **log** – Log modulation data to files? (boolean)

`gnuradio.digital.`**`soft_dec_table_generator`**(*soft_dec_gen*, *prec*, *Es=1*)

Builds a LUT that is a list of tuples. The tuple represents the
soft decisions for the constellation/bit mapping at any given
point in the complex space, (x,y).

The table is built to a precision specified by the ‘prec’
argument. There are (2x2)^prec samples in the sample space, so we
get the precision of 2^prec samples in both the real and imaginary
axes.

The space is represented where index 0 is the bottom left corner
and the maximum index is the upper left. The table index for a
surface space with 4 bits of precision looks like the following:

```
240 241 242 243 244 245 246 247 | 248 249 250 251 252 253 254 255
224 225 226 227 228 229 230 231 | 232 233 234 235 236 237 238 239
208 209 210 211 212 213 214 215 | 216 217 218 219 220 221 222 223
192 193 194 195 196 197 198 199 | 200 201 202 203 204 205 206 207
176 177 178 179 180 181 182 183 | 184 185 186 187 188 189 190 191
160 161 162 163 164 165 166 167 | 168 169 170 171 172 173 174 175
144 145 146 147 148 149 150 151 | 152 153 154 155 156 157 158 159
```

```
128 129 130 131 132 133 134 135 | 136 137 138 139 140 141 142 143
---------------------------------------------------------------
112 113 114 115 116 117 118 119 | 120 121 122 123 124 125 126 127
     96 97 98 99 100 101 102 103 | 104 105 106 107 108 109 110 111
     80 81 82 83 84 85 86 87 | 88 89 90 91 92 93 94 95
     64 65 66 67 68 69 70 71 | 72 73 74 75 76 77 78 79
     48 49 50 51 52 53 54 55 | 56 57 58 59 60 61 62 63
     32 33 34 35 36 37 38 39 | 40 41 42 43 44 45 46 47
     16 17 18 19 20 21 22 23 | 24 25 26 27 28 29 30 31
          0 1 2 3 4 5 6 7 | 8 9 10 11 12 13 14 15
```

We then calculate coordinates from -1 to 1 with 2^prec points for both the x and y axes. We then sample starting at (-1, -1) and move left to right on the x-axis and then move up a row on the y-axis. For every point in this sampled space, we calculate the soft decisions for the given constellation/mapping. This is done by passing in the function 'soft_dec_gen' as an argument to this function. This takes in the x/y coordinates and outputs the soft decisions. These soft decisions are stored into the list at the index from the above table as a tuple.

The function 'calc_from_table' takes in a point and reverses this operation. It converts the point from the coordinates (-1,-1) to (1,1) into an index value in the table and returns the tuple of soft decisions at that index.

Es is the maximum energy per symbol. This is passed to the function to provide the bounds when calling the generator function since they don't know how the constellation was normalized. Using the (maximum) energy per symbol for constellation allows us to provide any scaling of the constellation (normalized to sum to 1, normalized so the outside points sit on +/-1, etc.) but still calculate the soft decisions as we would given the full constellation.

`gnuradio.digital.`**`soft_dec_table`**(*constel*, *symbols*, *prec*, *npwr=1*)

Similar in nature to soft_dec_table_generator above. Instead, this takes in the constellation and symbol points along with the noise power estimate and uses calc_soft_dec (below) to generate the LUT.

Instead of assuming that the constellation is normalied (e.g., all points are between -1 and 1), this function calculates the min/max of both the real and imaginary axes and uses those when constructing the LUT. So when using this version of the LUT, the samples and the constellations must be working on the same magnitudes.

Because this uses the calc_soft_dec function, it can be quite a bit more expensive to generate the LUT, though it should be one-time work.

`gnuradio.digital.`**`calc_soft_dec_from_table`**(*sample*, *table*, *prec*, *Es=1.0*)

Takes in a complex sample and converts it from the coordinates (-1,-1) to (1,1) into an index value. The index value points to a location in the provided LUT 'table' and returns the soft decisions tuple at that index.

sample: the complex sample to calculate the soft decisions from.

table: the LUT.

prec: the precision used when generating the LUT.

Es: the energy per symbol. This is passed to the function to provide the bounds when calling the generator function since they don't know how the constellation was normalized. Using the (maximum) energy per symbol for constellation allows us to provide any scaling of the constellation (normalized to sum to 1, normalized so the outside points sit on +/-1, etc.) but still calculate the soft decisions as we would given the full constellation.

`gnuradio.digital.`**`calc_soft_dec`**(*sample*, *constel*, *symbols*, *npwr=1*)

This function takes in any conseallation and symbol symbol set (where symbols[i] is the set of bits at constellation point constel[i] and an estimate of the noise power and produces the soft decisions for the given sample.

If known, the noise power of the received sample may be passed in to this function as npwr.

This is an incredibly costly algorthm because it must calculate the Euclidean distance between the sample and all points in the constellation to build up its probability calculations. Conversely, it should work for any given constellation/symbol map.

The function returns a vector of k soft decisions. Decisions less than 0 are more likely to indicate a ‘0’ bit and decisions greater than 0 are more likely to indicate a ‘1’ bit.

`gnuradio.digital.`**`show_table`**(*table*)