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

gnuradio.fec

class gnuradio.fec.**cc_decoder**(**args*, ***kwargs*)

Convolutional Code Decoding class.

This class performs convolutional decoding via the Viterbi algorithm. While it is set up to take variable values for K, rate, and the polynomials, currently, the block is only capable of handling the following settings:

This is the well-known convolutional part of the Voyager code implemented in the CCSDS encoder.

The intent of having this FECAPI code classes fully parameterizable is to eventually allow it to take on generic settings, much like the cc_encoder class where the CCSDS settings would be a highly-optimized version of this.

The decoder is set up with a number of bits per frame in the constructor. When not being used in a tagged stream mode, this encoder will only process frames of the length provided here. If used in a tagged stream block, this setting becomes the maximum allowable frame size that the block may process.

The is a cc_mode_t that specifies how the convolutional encoder will behave and under what conditions.

A common convolutional encoder uses K=7, Rate=1/2, Polynomials=[109, 79]. This is the Voyager code from NASA:

class gnuradio.fec.**cc_encoder**(**args*, ***kwargs*)

Convolutional Code Encoding class.

This class performs convolutional encoding for unpacked bits for frames of a constant length. This class is general in its application of the convolutional encoding and allows us to specify the constraint length, the coding rate, and the polynomials used in the coding process.

The parameter sets the constraint length directly. We set the coding rate by setting to R given a desired rate of 1/R. That is, for a rate 1/2 coder, we would set to 2. And the polynomial is specified as a vector of integers, where each integer represents the coding polynomial for a different arm of the code. The number of polynomials given must be the same as the value .

The encoding object holds a shift register that takes in each bit from the input stream and then ANDs the shift register with each polynomial, and places the parity of the result into the output stream. The output stream is therefore also unpacked bits.

The encoder is set up with a number of bits per frame in the constructor. When not being used in a tagged stream mode, this encoder will only process frames of the length provided here. If used in a tagged stream block, this setting becomes the maximum allowable frame size that the block may process.

The is a cc_mode_t that specifies how the convolutional encoder will behave and under what conditions.

A common convolutional encoder uses K=7, Rate=1/2, Polynomials=[109, 79]. This is the Voyager code from NASA: Another encoder class is provided with gr-fec called the gr::fec::code::ccsds_encoder, which implements the above code that is more highly optimized for just those specific settings.

class gnuradio.fec.**ccsds_encoder**(**args*, ***kwargs*)

CCSDS Encoding class for convolutional encoding with rate 1/2, K=7, and polynomials [109, 79].

Uses Phil Karn's (KA9Q) implementation of the CCSDS encoder for rate 1/2, K=7, and CC polynomial [109, 79]. These are non-adjustable in this encoder. For an

adjustable CC encoder where we can set the rate, constraint length, and polynomial, see gr::fec::code::cc_encoder.

The encoder is set up with a number of bits per frame in the constructor. When not being used in a tagged stream mode, this encoder will only process frames of the length provided here. If used in a tagged stream block, this setting becomes the maximum allowable frame size that the block may process.

The is a cc_mode_t that specifies how the convolutional encoder will behave and under what conditions.

A common convolutional encoder uses K=7, Rate=1/2, Polynomials=[109, 79]. This is the Voyager code from NASA:

class gnuradio.fec.**dummy_decoder**(**args*, ***kwargs*)

Dummy Decoding class.

A dummy decoder class that simply passes the input to the output. It is meant to allow us to easily use the FEC API encoder and decoder blocks in an application with no coding.

class gnuradio.fec.**dummy_encoder**(**args*, ***kwargs*)

Dummy Encoding class.

A dummy encoder class that simply passes the input to the output. It is meant to allow us to easily use the FEC API encoder and decoder blocks in an application with no coding.

class gnuradio.fec.**ldpc_decoder**(**args*, ***kwargs*)

Proxy of C++ gr::fec::ldpc_decoder class.

class gnuradio.fec.**ldpc_encoder**(**args*, ***kwargs*)

Proxy of C++ gr::fec::ldpc_encoder class.

class gnuradio.fec.**repetition_decoder**(**args*, ***kwargs*)

Repetition Decoding class.

A repetition decoder class. This takes a majority vote, biased by the rate, and decides if the number of 1 bits > ap_prob, it is a 1; else, it is a 0.

class gnuradio.fec.**repetition_encoder**(**args*, ***kwargs*)

Repetition Encoding class.

A repetition encoder class that repeats each input bit times. To decode, take a majority vote over the number of repetitions.

class gnuradio.fec.**tpc_decoder**(**args*, ***kwargs*)

Proxy of C++ gr::fec::tpc_decoder class.

class gnuradio.fec.**tpc_encoder**(**args*, ***kwargs*)

Proxy of C++ gr::fec::tpc_encoder class.

class gnuradio.fec.**bercurve_generator**(*encoder_list*, *decoder_list*, *esno=array([0.*, *0.25*, *0.5*, *0.75*, *1.*, *1.25*, *1.5*, *1.75*, *2.*, *2.25*, *2.5*, *2.75])*, *samp_rate=3200000*, *threading='capillary'*, *puncpat='11'*, *seed=0*)

gnuradio.fec.**bitreverse**(*mint*)

gnuradio.fec.**bitflip**(*mint*, *bitflip_lut*, *index*, *csize*)

gnuradio.fec.**read_bitlist**(*bitlist*)

gnuradio.fec.**read_big_bitlist**(*bitlist*)

gnuradio.fec.**generate_symmetries**(*symlist*)

class gnuradio.fec.**capillary_threaded_decoder**(*decoder_list_0*, *input_size*, *output_size*)

class gnuradio.fec.**capillary_threaded_encoder**(*encoder_list_0*, *input_size=1*, *output_size=1*)

class gnuradio.fec.**extended_async_encoder**(*encoder_obj_list*, *puncpat=None*)

class gnuradio.fec.**extended_decoder**(*decoder_obj_list*, *threading*, *ann=None*, *puncpat='11'*, *integration_period=10000*, *flush=None*, *rotator=None*)

class gnuradio.fec.**extended_encoder**(*encoder_obj_list*, *threading*, *puncpat=None*)

class gnuradio.fec.**extended_tagged_decoder**(*decoder_obj_list*, *ann=None*, *puncpat='11'*, *integration_period=10000*, *flush=None*, *rotator=None*, *lentagname=None*, *mtu=1500*)

class gnuradio.fec.**extended_tagged_encoder**(*encoder_obj_list*, *puncpat=None*, *lentagname=None*, *mtu=1500*)

class gnuradio.fec.**fec_test**(*generic_encoder=0*, *generic_decoder=0*, *esno=0*, *samp_rate=3200000*, *threading='capillary'*, *puncpat='11'*, *seed=0*)

class gnuradio.fec.**threaded_decoder**(*decoder_list_0*, *input_size*, *output_size*)

class gnuradio.fec.**threaded_encoder**(*encoder_list_0*, *input_size*, *output_size*)