Task 1

An underwater habitat controls a remotely-operated submarine by sending the coordinates that the submarine has to traverse in sequence. The 3-dimensional coordinates are organized into triplets, as xxx.xxx,yyy.yyy,zzz.zzz; . Each digit and separating characters ("," and ";") are encoded as an 8-bit ASCII character. Triplets are concatenated and sent sequentially. This datastream is the input to the modulating system of the transmitter.

The transmitter first organizes the input stream into packets. Each packet consists of a preamble, header, data, CRC-16 hash, and a postamble. The frame structure is illustrated in Figure 1.

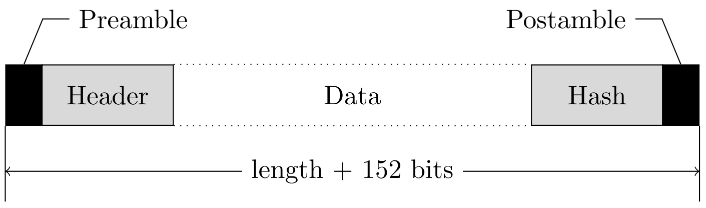

Figure 1: Packet structure

Preamble is a predefined 64-bit sequence `0xa5a5a5a5a5a5a5a5`. Postamble is a bitwise-inverse of the preamble. Header is an 8-bit word containing byte-length of the data section in the packet. The shortest size of the data section is 64 bits, while the largest is 248 bits. The data section is always byte-aligned; i.e. its length is always a multiple of 8 bits.

All the words are stored in packets in big endian notation. The packets are forwarded further in the transmitter chain sequentially, without a guard interval.

The binary sequence of the generated packets is brought to a QPSK modulator. Its constellation is represented by Figure 2.

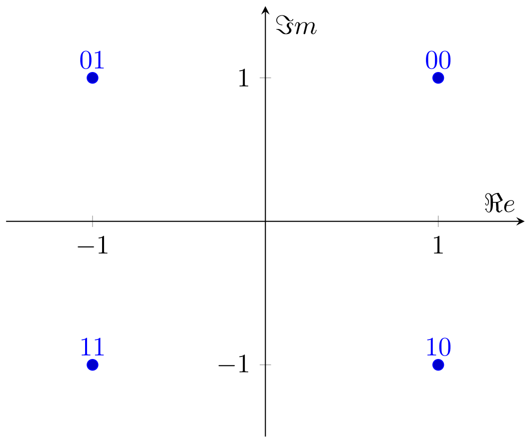

Figure 2: QPSK modulator constellation

The QPSK-modulated symbols are transposed to carrier frequency $f_c = 36\,\text{kHz}$ and transmitted by a hydrophone. Transmitter data-rate is 9.6 kbit/s.

At the receiving end, the receiver aboard a submarine is depicted by a block diagram in Figure 3.

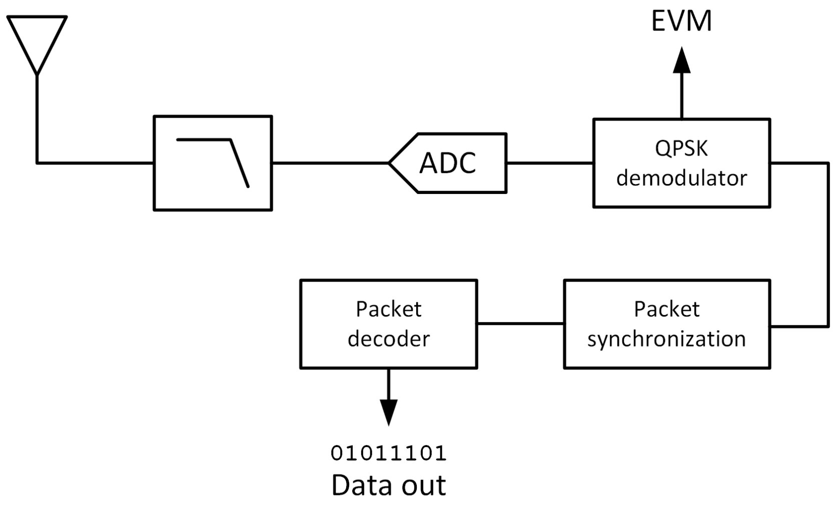

Figure 3: Receiver diagram in Task 1

The ADC of the receiving hydrophone operates at a sampling frequency of 192 kSps. An ideal low-pass filter has a cut-off frequency of 96 kHz.

You are required to implement the digital processing system of the receiver, to demodulate and decode information from the received ultrasonic signal.

Input data

The single_carrier.raw file is given. It contains the digitized signal from the ADC, recorded in 32-bit float format.

Implementation

Your solution is a C or C++ code implementing the receiver's digital processing chain. You are required to submit at least two files:

radio.c / cpp Implements all the routines described further in the text.

main.c / cpp Loads the input data, calls the routines from radio.c / cpp, and outputs decoded data to stdout in the original format.

Should your solution use more source files, you must provide a Makefile that compiles the rest of the source files.

Your radio.c / cpp file must implement the following set of functions:

complex *frequency_shift(double *input, double fc, double fs, int N)
Transposes the signal provided in input from centre frequency fc to baseband. N is the length of the data stream, expressed as a number of double words. fs is the sampling frequency.

double qpsk_demodulator(complex symbol, double constellation_offset, char *decoded_symbol)
Decodes a QPSK symbol symbol to a word, stored in an 8-bit format to decoded_symbol. The function outputs the EVM of the symbol.

char *bitstream_to_bytestream(char *bitstream, int length)
Converts length 8-bits words, as output by the previous function, into a bytestream. A pointer to the beginning of the bytestream is returned.

void frame_sync(char **bytestream, int length)
Moves the pointer *bytestream to the beginning of the first detected frame. length is the length of the bytestream.

void frame_decoder(char *bytestream, char **data)
Decodes the data from a single frame that starts at the pointer *bytestream. Data is stored to *data.

You are provided with some helper files:

radio.h

api.h
An API containing some types, functions and constants you may find useful.

api.o
The compiled API described in api.h.

Task 2

Along the submarine's trajectory information, the habitat and the submarine exchange location data on debris scattered around the sea. This data is transmitted in a separate data stream from the trajectory coordinates, but has the same triplet format, as the data from Task 1. The data is packaged in the same manner as in Task 1. The resulting bitstream is then spread into 4 QPSK subcarriers, as depicted in Figure 4.

The obtained QPSK carriers are multiplexed into OFDM that consists of a total of 10 subcarriers. These are organized as depicted by DFT spectrum in Figure 5.

Subcarrier spacing is $\Delta f = 1500\,\text{Hz}$. The graph is centred to frequency $f_c = 36\,\text{kHz}$, corresponding to the central frequency of the OFDM carrier. The subcarrier at bin 0, depicted by a dashed line, containes trajectory coordinate stream from Task 1. Subcarriers depicted by dotted lines contain the added data stream. Subcarriers depicted by a solid line are pilots. OFDM symbols have no cyclic prefix.

The receiver code from Task 1 has to be modified to work with an OFDM carrier, and decode the additional data stream. Block diagram of the receiver is presented in Figure 6.

Input data

The ofdm_carrier.raw file is given. It contains the digitized signal from the ADC, recorded in 32-bit float format.

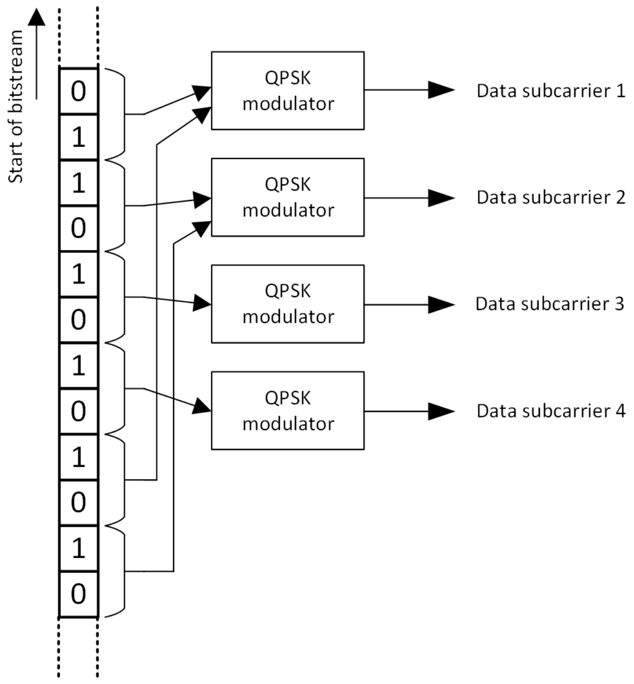

Figure 4: Bitstream spreading mechanism

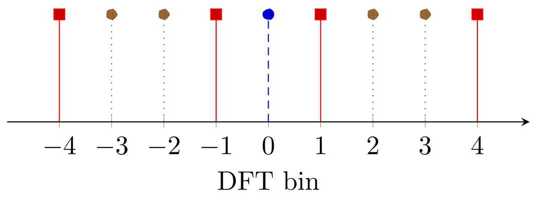

Figure 5: OFDM carrier DFT spectrum

Implementation

You should update your radio.c / cpp and main.c / cpp to implement the new features required for Task 2. Your radio.c / cpp file should implement the following additional functions:

double *ofdm_demodulator(complex *input, int *carrier_idx, int carrier_no, char **data)

Demultiplexes the additional data stream from the OFDM carrier given in input, and stores it into *data. The input signal is in baseband in the original sample rate. carrier_idx is a list of DFT bins that contain the additional data stream. carrier_no is the length of carrier_idx array.

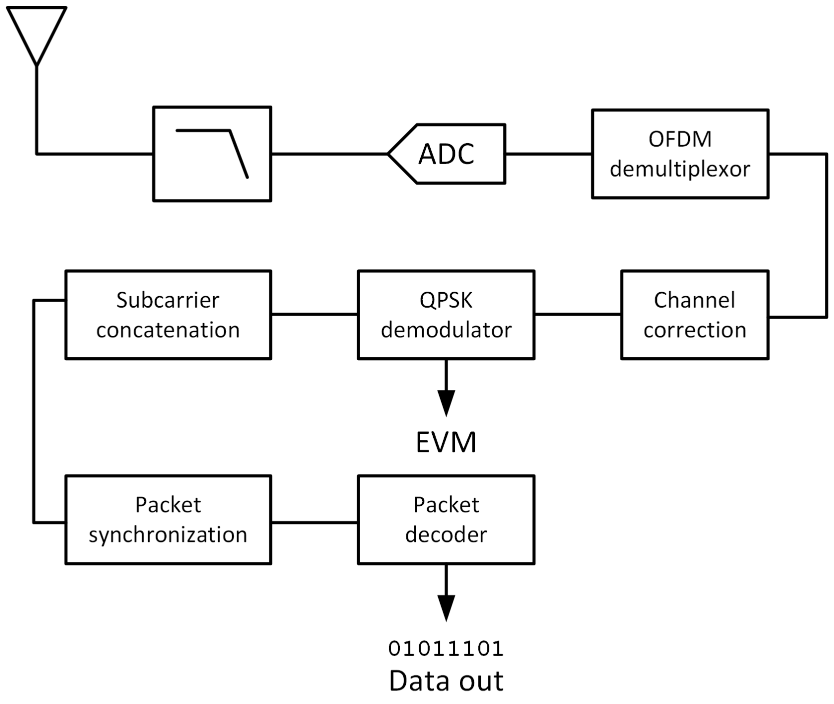

Figure 6: Receiver diagram in Task 2

Task 3

An algorithm for calculating the CRC-16 hash of an input stream should be added to solutions for both tasks. CRC-16 verification should first be implemented as a separate function, and then integrated into the receiver chain. The modification is presented in Figure 7.

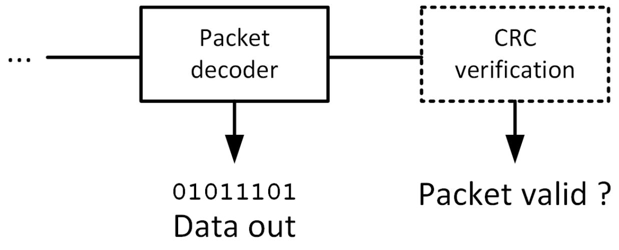

Figure 7: CRC check in receivers

The CRC-16 hash in data frames is calculated using the CRC-CCITT algorithm, with an initial register state `0xffff`.

Implementation

Your `radio.c` / `cpp` code must implement the following additional functions:

char crc16_check(char* bitstream, int length)
Returns CRC-16 hash of the given `bitstream`, of length `length`.

bool frame_decoder(char* bytestream, char data)**
Decodes the data from a single frame that starts at the pointer `*bytestream`. Data is stored to `*data`. The return value specifies if the integrity of the decoded frame is preserved.

Documentation

Provide documentation that describes your solution for the receiving chain. Back your implementation by describing the mathematical background of your processing chain. Document the datasets you used to evaluate your solution.

The documentation will not be graded, but it can help mentors in understanding your approach and validating the theory behind your solution.