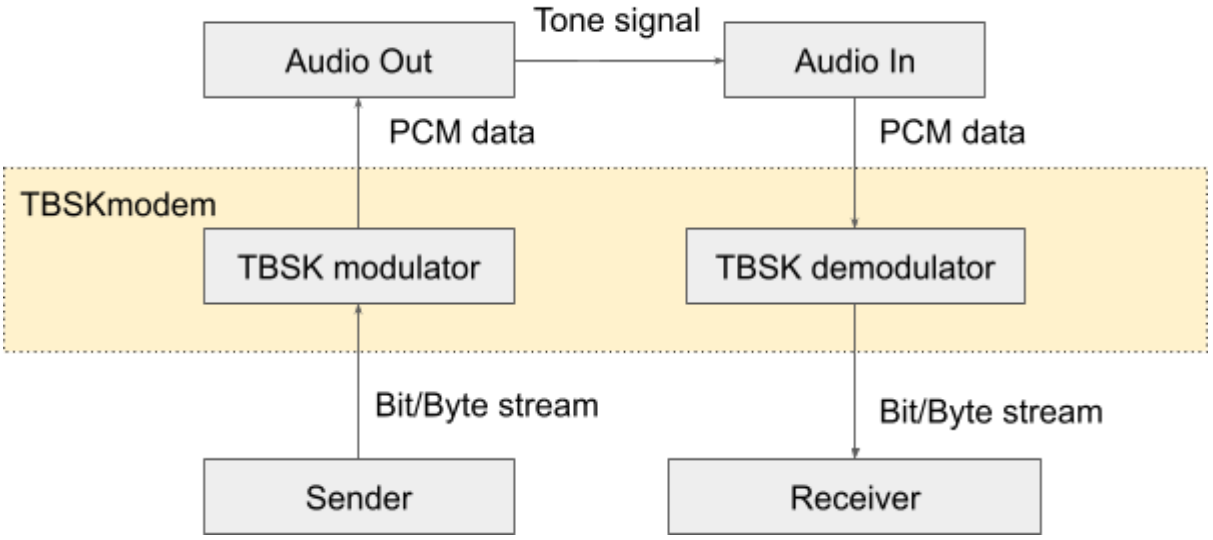
TBSKmodem Rev4

https://nyatla.jp/

TBSKmodem is a short-range ultrasonic communication system designed for air-based communication. It operates without the need for dedicated communication equipment and can function with smartphones, computer audio devices, and analog I/O on microcontrollers. The practical maximum transmission speed is 1 kbps, and it does not have error correction or detection capabilities.



This document provides detailed explanations about TBSKmodem, including modulation method, signal format, multicarrier modulation procedure, and implementation.。

Overview

TBSKmodem（Tone Binary Shift Keying Modem）is designed for low-bitrate short-range digital transmission using sound waves modulated by the TBSK modulation method. It offers a maximum transmission speed of 1 kbps, with 1 bit per symbol. Typical bit rates include 80 bps, 160 bps, 320 bps, 480 bps, and 960 bps.

In the demodulation process, it employs delayed detection of the amplitude signal and direct symbol recovery from correlation values.

The carrier signal consists of distinctive tone signals with arbitrary frequency bandwidth, conveying information through differential phase modulation. This design is intended to work in environments with unknown or unstable frequency attenuation characteristics and can be used in various scenarios.

The tone signal can have flexible amplitude signal shapes, such as single-frequency sine waves, square waves, or composite waves. Typically, these basic signals are spread over a wide bandwidth, but they can also be used directly when necessary.

The amplitude signal occupies both spatial and frequency bandwidth within the communication channel, with no multiplexing other than time division assumed.

TBSKmodem is a connectionless, error-detection and correction-free line. Challenges like collision control and connections are expected to be addressed at higher layers.

Signal type

This section provides an explanation of TBSKmodem's carrier signal type, symbol type, modulation, and demodulation methods.

Carrier Signal Type

The carrier signal consists of unit tone signals that are separated into periods of T seconds. The distinctive tone signal used is not a constant value.

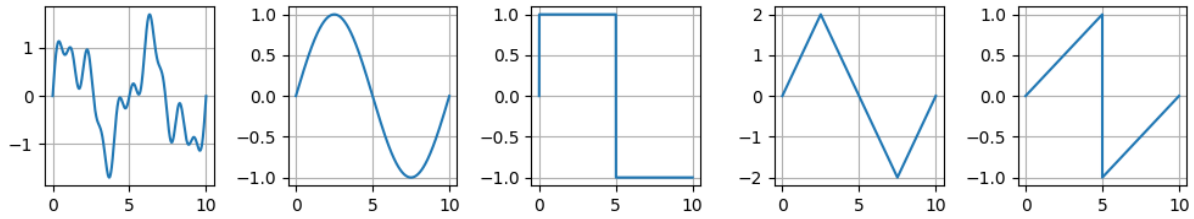


Figure: Example of a Unit Tone Signal

Unit tone signals are continuous and form the TBSK carrier signal. The shape of the unit tone signal is arbitrary, but within a single TBSK carrier, the same unit tone signal must be used throughout.

Modulation Method

In TBSKmodem, two types of symbols are defined: the unit tone signal and the unit tone signal with its phase inverted. The original signal is represented as symbol P, and the inverted signal is symbol N. The length of symbols is the same as that of the unit tone signal.

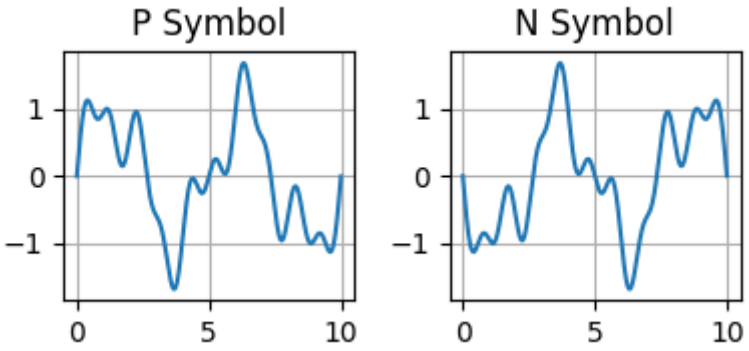


Figure: Example of P and N Symbols

The symbol value [n] is a differential value. If adjacent symbols [n-1] and [n] are the same, it's represented as 1; if they are different, it's represented as 0. Symbol values are determined based on the relationship between adjacent unit tone signals. Even with different carrier signals using different unit tone signals, the same symbol value can be obtained as long as the period of the unit tone signals is the same.

These symbols transmit 1 bit per symbol. The baud rate is determined by the reciprocal of the length T of the amplitude signal. For example, if the length of the unit tone signal is 10 ms, the baud rate is 100 Baud.

TBSK Frame

To transmit digital values, TBSK frames are used. The TBSK frame is a data format employed by TBSKmodem for signal detection and payload storage.

The TBSK frame consists of a preamble section for signal detection and synchronization and a payload section for storing the bit sequence. These two fields are contiguous.

The frame layout is defined with a cycle value of 4.

Table: TBSK Frame Layout

Field Name	TBSK Frame	
	Preamble	Payload
Symbol	14symbols	0 symbols or more
Symbol value	13bits	0 bits or more

The TBSK frame does not have an explicit termination. The timing for demodulating the payload is determined by the time defined in the upper layer or when the absolute correlation value falls below a threshold for a continuous duration.

In adjacent fields, symbol value sequences can be independently defined, but symbol sequences are influenced by the termination of the preceding field. Symbol sequences are differential values and are used to reference symbols from adjacent fields.

Preamble Field

This is a symbol field indicating the starting point of the TBSK frame. The preamble's length is influenced by the parameter "cycle."

The preamble takes n bits of symbol values and expands them into n+1 symbols. The increase of one symbol is for the conversion to differential values.

The Python code for generating the symbol sequence is defined as follows.

```
b=[1]*cycle
c=[i%2 for i in range(cycle)]
d=[(1+c[-1])%2,(1+c[-1])%2,c[-1],]
preamble=[0,1]+b+[1]+c+d
```

The symbol sequence and symbol value sequence for the preamble with a default cycle value of 4 are provided in the following table.

Table: Preamble Layout of Symbol Values (cycle=4)

No.	-1	0	1	2	3	4	5	6	7	8	9	10	11	12
Sub field name	Prefix		Positive					Negative					Sync	
Symbol	N	P	P	P	P	P	P	N	P	N	P	N	N	P
Symbol value		0	1	1	1	1	1	0	0	0	0	0	1	0

Symbol array(cycle=4) = [N,P,P,P,P,P,P,N,P,N,P,N,N,P]
Symbol value array(cycle=4) = [0,1,1,1,1,1,1,0,0,0,0,0,1,0]

The symbol values consist of four subfields:

- Prefix - Training values before the signal output. Not used for detection.
- Positive - Continuous output of in-phase signals. Used for detection and power measurement.
- Negative - Continuous output of out-of-phase signals. Used for detection and power measurement.
- Sync - Symbol synchronization position. Used to determine the synchronization position of symbols.

The Positive subfield and Negative subfield vary in accordance with the cycle parameter.

Payload Field

The payload field is a variable-length bit sequence. Starting from the beginning of the bit sequence, each bit value is directly converted into a symbol value. In the case of byte values, the bit values are taken from the most significant bit to the least significant bit.

The payload section is concatenated with the preamble section. The 0th symbol of the payload is a differential value with respect to the last symbol value of the preceding preamble section. The symbol sequence of the payload section is distinct based on the last symbol of the preamble section.

Below is an illustration of how an 8-bit bit stream "b00101011" (0x2b) is concatenated with a preamble section with a cycle value of 4. Please note that the preamble with cycle=4 has its last symbol as "P."

Table: Payload Layout

	Preamble	Payload							
bit/sym No	-	0	1	2	3	4	5	6	7
bit value	-	0	0	1	0	1	0	1	1
symbol value	0111110000010	0	0	1	0	1	0	1	1
symbol	NPPPPPPNPNPNNP	N	P	P	N	N	P	P	P

The TBSK modem is at 1 bit per symbol, so the transmission speed of the payload section is the same as the baud rate.

Warm-Up/Cool-Down

To output to the audio device, warm-up and cool-down signals are added. The warm-up signal is inserted before the modulated TBSK frame of amplitude signals, and the cool-down signal is added at the end.

These signals have the effect of stabilizing the acoustic device before transmitting the modulated carrier wave. They also serve as a measure to address issues in audio systems integrated into information equipment where the beginning and end of audio signals are not played back correctly.

Table: Layout of the Entire Signal

Signal Type	Uncorrelated Signal	TBSK Modulated Signal		Uncorrelated Signal
Signal name	Warm-Up	TBSK Frame		Cool-Down
		Preamble	Payload	
Length	0[ms] or more	T*(cycle*2+6) [ms]	N*T [ms]	0[ms] or more

Signals suitable for warm-up and cool-down are those in which the correlation values of waveforms, segmented at symbol-length intervals across the entire signal, continuously approach or become positive near zero.

Warm-up and cool-down signals should not resemble the preamble and be of a similar amplitude signal. Signals that resemble the preamble signal waveform are those for which the absolute value of correlation becomes significant at the intervals of unit-tone signals. Such signals are at a higher risk of being falsely detected as a preamble.

Empirically, a period of about 30 ms is appropriate for warm-up and cool-down signals.

When transmitting the completed signal, it is important to pay attention to the device's sampling rate and adjust the output of unit-tone signals to the intended duration.

Modulation and Demodulation

In this chapter, we will set specific parameters and explain the modulation and demodulation procedures.

The signal specifications are as follows: the preamble parameter is set with a cycle of 4, a baud rate of 80 Baud, and the shape of the baseband signal is a sawtooth wave. In addition to the data part, a 30ms warm-up and cool-down signal will be included. The output data format will be a sampling rate of 8000Hz with 8-bit signed PCM data.

With the above specifications, we will create an amplitude-modulated signal to transmit the 4-byte ASCII code "TBSK.

Modulation

In the modulation process, we convert a bit value sequence into an amplitude signal. This is done through the following steps:

1. Creating Tone Signals: Generate tone signals.
2. Creating TBSK Frame: Build TBSK frames.
3. Creating Modulated Signals: Generate the modulated signal using the symbol sequence and tone signals.

To make it an audio signal that can be played on audio devices, we add the following steps:

4. Adding Warm-Up/Cool-Down Signals: Incorporate warm-up and cool-down signals.

Creating Tone Signals

If the audio source has a sampling rate of 8000Hz and you are transmitting symbols at 80 Baud, then the number of samples (ticks) assigned to each symbol would be 8000/80 = 100 ticks.

In this context, you would create a sawtooth wave with a period of 100 ticks, meaning one cycle occurs over 100 ticks.

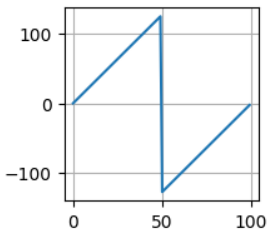


Figure:Sawtooth wave

The number of ticks for a unit tone signal is empirically considered appropriate to be between 50 and 100 ticks. If it's too short, it degrades the quality of correlation values, and if it's too long, it increases the computational cost of correlation. Typical values for each sampling frequency are as follows.

Table: Typical values for each sampling frequency

Frequency(Hz)	unit tone		Baud	bps
	Tick size(ticks)	time(ms)		
8000	100	12.50	80	80
16000	100	6.25	160	160
24000	100	1000/240 ≒ 4.17	240	240
32000	100	1000/320 ≒ 3.13	320	320
48000	100	1000/480 ≒ 2.08	480	480
8000	50	6.25	160	160
16000	50	1000/320 ≒ 3.13	320	320
24000	50	1000/480 ≒ 2.08	480	480
32000	50	1000/640 ≒ 1.56	640	640
48000	50	1000/960 ≒ 1.04	960	960

Creating TBSK Frame

This TBSK frame, as per the specifications, consists of 14 symbols in the preamble section and 32 symbols in the payload section, for a total of 46 symbols.

First, let's create the symbol sequence for the preamble section. According to the specifications, the symbol sequence for the preamble section with a cycle of 4 is [NPPPPPPNPNPNNP].

Next, let's create the symbol sequence for the payload. We will convert the characters "TBSK" into individual ASCII code bit values and then, considering that the preamble section ends with "P," transform the bit sequence into a sequence of differential values.

Table: Payload layout

Charactor	T	B	S	K
ASCII code	86	66	8.	75
bits	01010100	01000010	01010011	01001011

Symbols	NNPPNNPN	PPNPNPPN	PPNNPNNN	PPNPPNNN
Joined Symbols	NNPPNNPNPPNPNPPNPPNNPNNNPPNPPNNN			

Combining the two symbol sequences we created completes the symbol sequence of the TBSK frame.

Table:.TBSK Frame layout

Field name	Preamble	Payload
Symbols	NPPPPPPNPNPNNP	NNPPNNPNPPNPNPPNPPNNPNNNPPNPPNNN
Joined Symbols	NPPPPPPNPNPNNPNNPPNNPNPPNPNPPNPPNNPNNNPPNPPNNN	

Creating Modulated Signals

The modulation signal of TBSK is created by concatenating unit tone signals in the forward or reverse direction based on the values of the symbol sequence. Starting from the beginning of the symbol sequence in the TBSK frame, if the symbol is "P," the unit tone signal remains as is; if it's "N," the unit tone signal is reversed and then concatenated sequentially.

As a result, the 46-symbol sequence expands into 4600 ticks of PCM data.

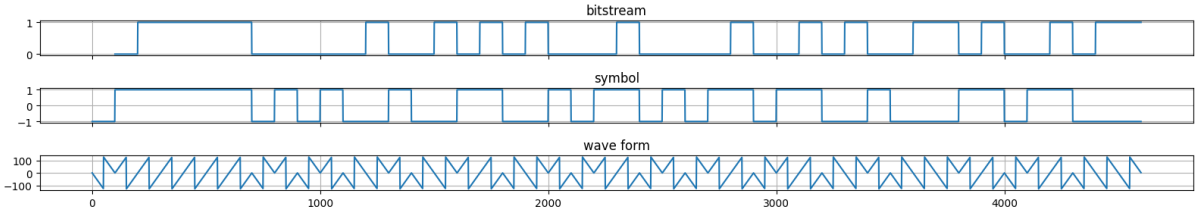


Figure: Modulated PCM

Adding Warm-Up/Cool-Down Signals

Warm-up and cool-down signals are desired to be uncorrelated. A simple method for their generation involves using random values. Here, we generate them using XorShift. We produce 240 UINT8 values, convert them to 'signd' format, and adjust them to center around the zero point.

The signal can be generated using the following Python code.

```
""" https://ja.wikipedia.org/wiki/Xorshift
"""
def xorshift31(size=10,seed=299,skip=0):
    ret=[]
    y=seed
    for i in range(size):
      y = y ^ (y << 13)
      y = y ^ (y >> 17)
      y = y ^ (y << 5)
      y=y & 0x7fffffff
      ret.append(y)
    return ret
v=[(i%256)-128 for i in xorshift31(size=240)]
```

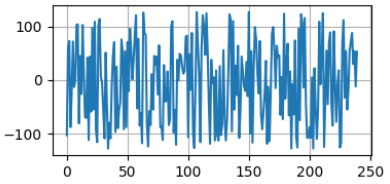


Figure: Uncorrelated Signal

Completed TBSK modem signal

By adding warm-up and cool-down signals before and after the modulated signal, the TBSK modem signal is completed. The resulting signal has a total of 5080 ticks, with a sampling rate of 8000 Hz, and a duration of 635 milliseconds.

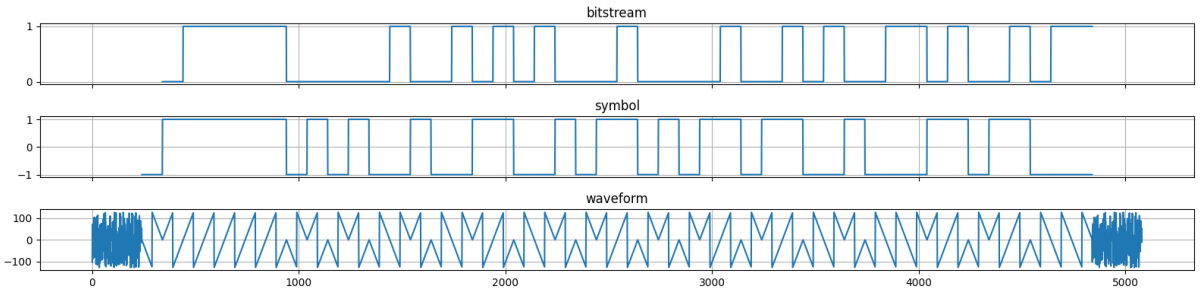


Figure: Symbols, Modulated Signal

Demodulation

In demodulation, the amplitude signal is converted into a bit value sequence using the following steps:

1. Correlation Value Analysis
2. Detection of Positive Subfields
3. Detection of Negative Subfields
4. Determination of Synchronization Position
5. Evaluation of the Preamble
6. Acquisition of Payload
7. Estimation of Termination

While it is desirable to perform these processes sequentially, for simplicity, we will describe a method for processing recorded amplitude signals. The source amplitude signal for demodulation is the one created during modulation.

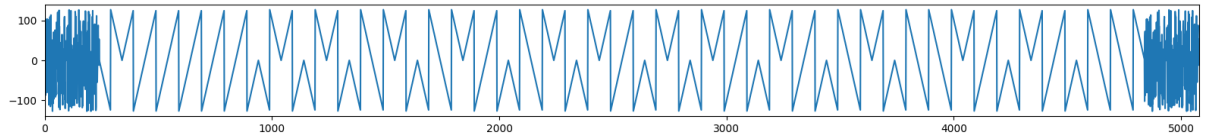


Figure: Source signal

Correlation Value Analysis

To determine symbols from the amplitude signal, you create a delayed signal B by shifting the amplitude signal (A) by the symbol length (100 ticks). Then, you calculate the correlation values continuously over the entire amplitude signal for a duration equal to the symbol length.

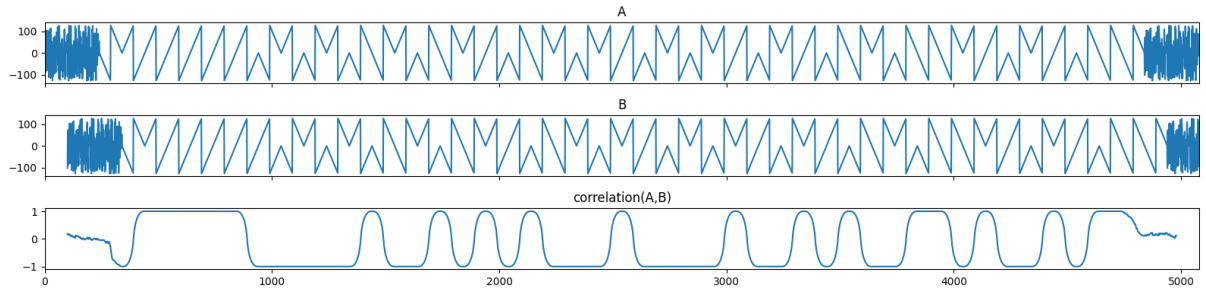


Figure: Amplitude signal and Correlation

The calculated correlation values are time series values within the range of [-1, 1].

Detection of Positive Subfields

Starting from the beginning of the amplitude signal, search for consecutive high correlation values. This corresponds to the portion of the Positive Subfield.

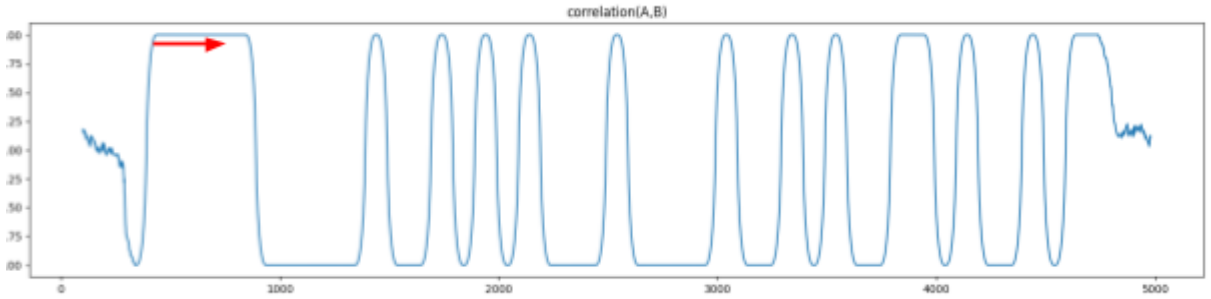


Figure: Positive subfield

Detection of Negative Subfields

Next, search for the starting point of the Negative Subfield.
If a Negative Subfield follows the Positive Subfield, the correlation values will significantly drop with a noticeable gap. When this gap exceeds the threshold, it is determined that the Negative Subfield is present.

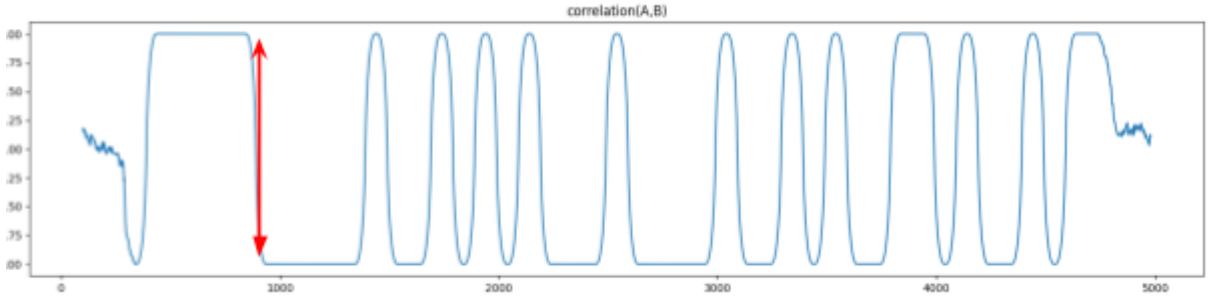


Figure: Negative subfield

Determination of Synchronization Position

Upon reaching the Sync Subfield that follows the Negative Subfield, the correlation values become positive, forming a peak, and then turn negative again. This peak represents the edge of the symbol boundary and serves as the synchronization position of the signal.

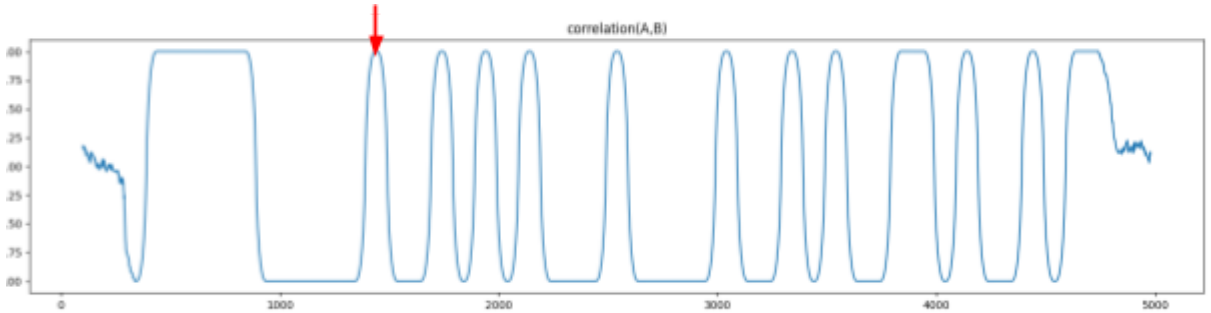


Figure: Sync subfield

Evaluation of the Preamble

Based on the synchronization position, evaluate the quality of the Positive Subfield and Negative Subfield. In each subfield interval, confirm that the correlation values remain within the specified range and for the designated duration. If the quality is good, proceed to acquire the payload by considering the signal to be present. If the quality is not satisfactory, restart the detection process from the Positive Field.

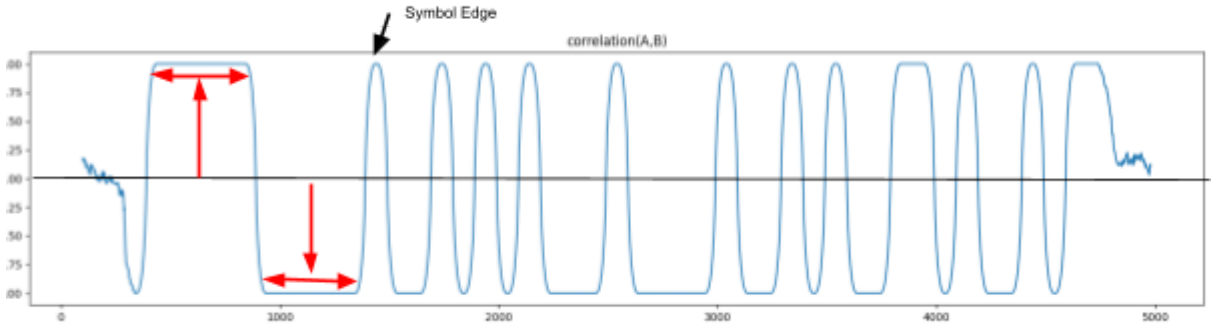


Figure: Preamble evaluation

Acquisition of Payload

Using the synchronization position as a reference, by sampling the correlation values at symbol length intervals following the preamble, you can obtain symbol values from these correlation values. These symbol values can be segmented into 8-bit units and then used to reconstruct ASCII codes.

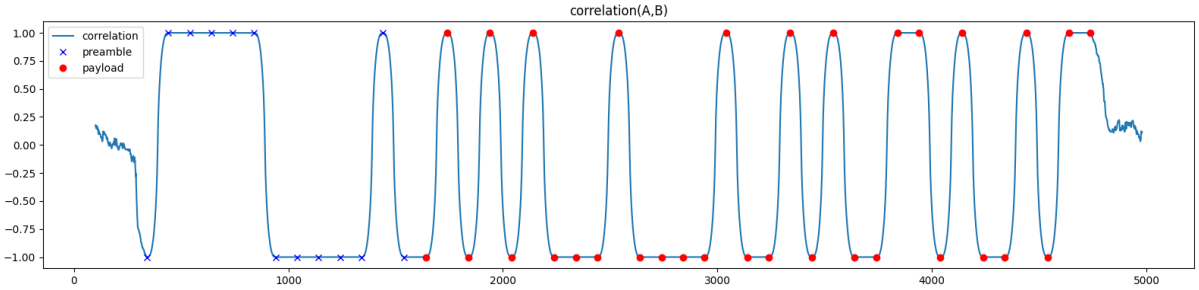


Figure: Symbol position

In audio devices that are not finely tuned, there may be slight frequency offsets or variations in the AD/DA conversion process, and when transmission extends over a long duration (typically around 10k ticks or more), symbol boundaries may no longer align. To address this, it becomes necessary to adjust the synchronization position during the reception of the payload.
When the symbol values of the payload invert, it results in a gradient in the correlation values. At this point, you can track the synchronization position by shifting it towards the higher correlation values.
It's important to note that this method is only meaningful for signals with symbol inversions. To make it more effective, encoding may be required at a higher-layer level.

Estimation of Termination

When the amplitude signal is finite, the endpoint can be determined by reading symbols until the termination is reached. If the signal is infinite, the endpoint is assumed to be reached when the absolute value of the correlation falls below a certain threshold.

Using uncorrelated signals in the cooldown signal can expedite the determination of the endpoint; however, it is susceptible to noise. Ideally, the endpoint should be defined at the higher protocol level.

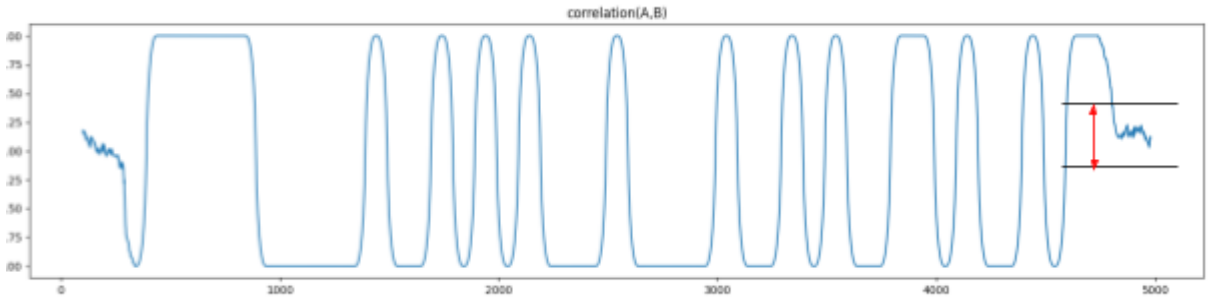


Figure: Termination

Performance Evaluation

This is a subjective assessment.

Communication Speed

Communication speed generally yields good results when a single-tone signal is constructed in 100 ticks or more. In a stable communication environment, normal communication was confirmed for up to 50 ticks.

With a sampling rate of 8 kHz, you can expect communication speeds of 80 bps, while at 16 kHz, 160 bps can be achieved. In the case of external microphone and speaker communication, a setting of 48 kHz/50 ticks allowed successful communication at a rate of 960 bps over a distance of 60 cm.

Communication Distance

For a communication rate of 160 bps, a communication distance of several meters is possible. This depends on the reproducibility of the acoustic devices and the communication environment. In a room with a lot of echoes, communication is limited to a few meters. A stable connection typically exists within 1 meter.

Environmental Characteristics

Communication was possible using built-in microphones, speakers, external speakers, and devices on PCs and smartphones. Data transfer via compressed audio communication, relayed through internet-based voice communication apps, was also achievable.

For embedded devices, standalone communication using piezoelectric buzzers is possible, but appropriate analog circuitry is required for reception.

Noise Characteristics

In the presence of spatial noise in the communication channel, the performance is generally equivalent to regular PSK communication. Some level of noise in certain frequency bands is tolerable. With spectrum-spread tone signals, even if some frequency bands are lost during transmission, it does not pose a problem.

Error Characteristics

For acoustic devices equipped with music recording and playback functions, errors do not occur with noise at the level of ambient household sounds. Random noise can result in bit-flip errors, which are sometimes correctable. However, burst errors lead to a loss of synchronization, causing subsequent data to be consistently destroyed.

Implementation

Illustrating libraries designed for various programming languages that implement the TBSKmodem specifications and their features. All libraries are compatible with the communication specifications and can modulate and demodulate signals interchangeably.

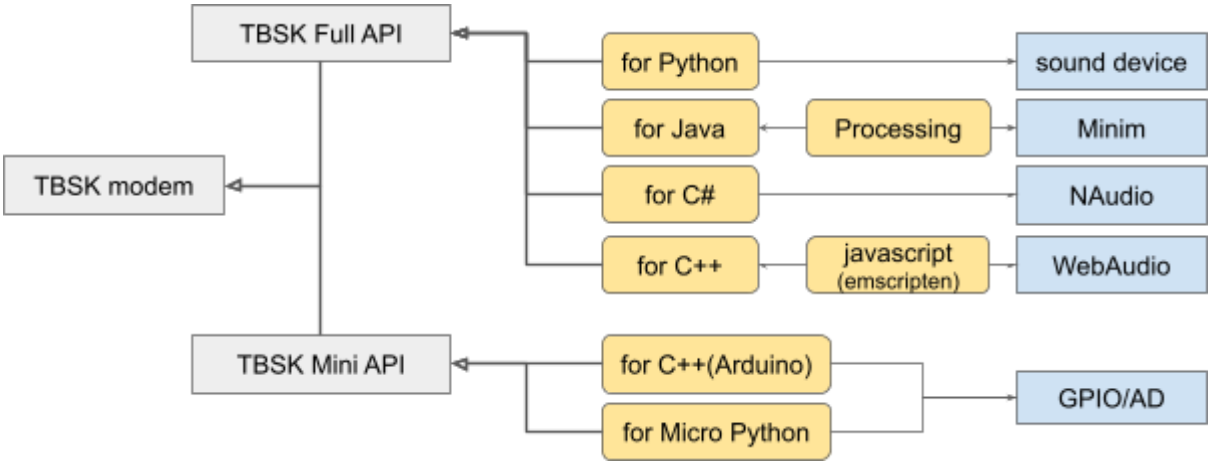


図.TBSKmodem libraries

Core Implementation

The core implementation is a set of code that includes the implementation of complex modulation operations. There are two different lineages with distinct design philosophies: Full and Mini, each with implementations in multiple languages. Full is designed for systems with ample memory and computational power, while Mini is designed for systems with limited memory and computational resources, utilizing only integer arithmetic. Both implementations do not rely on external libraries other than standard libraries.

Python

Full lineage. It serves as the reference implementation for the Full lineage.
Anaconda environment is recommended for the development setup.
The sounddevice library can be used for audio processing.
https://github.com/nyatla/TBSKmodem

Java

Full lineage. It serves as the reference implementation for managed languages. The API follows the Python version.
This implementation is for arithmetic operations only and does not support audio libraries.
https://github.com/nyatla/TBSKmodemJava

C#

Full lineage. It is a pure C# implementation. The API follows the Java version.
The NAudio library can be used for audio processing.
https://github.com/nyatla/TBSKmodemCS

C++

Full lineage. It is a C++ implementation. The API follows the Java version.
This implementation is for arithmetic operations only and does not support audio libraries.
Build environments are available for Visual Studio, Linux, and Emscripten.
https://github.com/nyatla/TBSKmodemCpp

Micro

Mini lineage. There are implementations in both C++ and MicroPython. It uses square wave carrier signals. The API is synchronous and differs from other implementations.

The MicroPython implementation is for modulation only.
The C++ implementation can be used in the Arduino environment, and it does not use a memory management mechanism.
Both modulation and demodulation are possible. GPIO is used for output, and AD pins are used for input.
It has been verified to run on Raspberry Pi Pico.
https://github.com/nyatla/TBSKmodemMicro

Wrapper Libraries

These libraries wrap the core implementation and provide an environment-specific API.

Processing

This implementation wraps the Java version. It offers a standard Serial-like API for Processing and a dedicated ModemAPI.
The minim library is used for audio processing.
https://github.com/nyatla/TBSKmodem-for-Processing

JavaScript

This implementation compiles the C++ version with Emscripten to create a JavaScript API.
It provides a Websocket-like API, is standalone, and supports npm.
WebRTC, wasm, and WebAudio are used for output.
https://github.com/nyatla/TBSKmodemJS

Live Demos

This is a live demo using the JavaScript version.
It works in modern browsers such as Chrome, Edge, and Safari.

https://nyatla.jp/tbskmodem/