EE 595A: Special Topics in Communication
Agricultural and Urban IoT Networked Systems

Spring 2024

Project 1: LoRa Packet Decoding

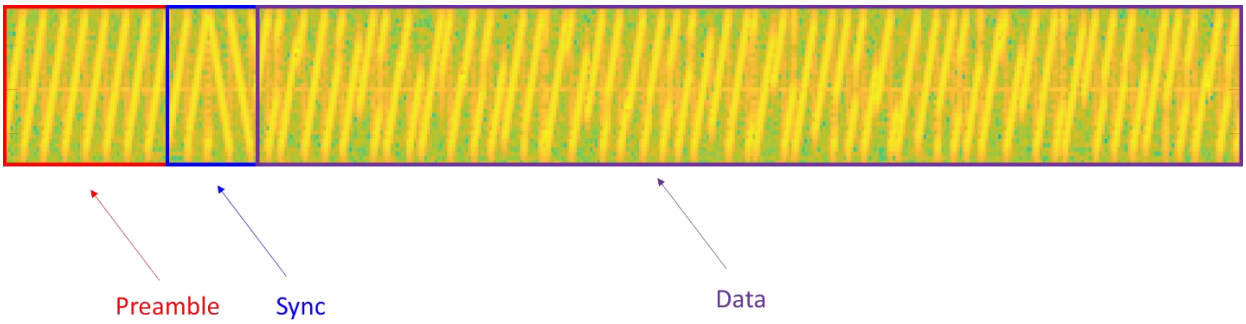
Deadline: May 01, 2024. 11:59 PM PST

Questions? Canvas Discussion => Canvas Message => Office Hours

1. LoRa

LoRa is a popular Low-Power Wide-Area Network (LP-WAN) technology that agricultural and urban IoT deployments leverage to connect large number of low power devices at a lower throughput. LoRa adopts the Chirp Spread Spectrum (CSS) modulation which makes it apt to operate in ISM band where existing occupants operate in FSK mode. In this project, you will be developing a decoder for this technology based on the things learned in class. [1]

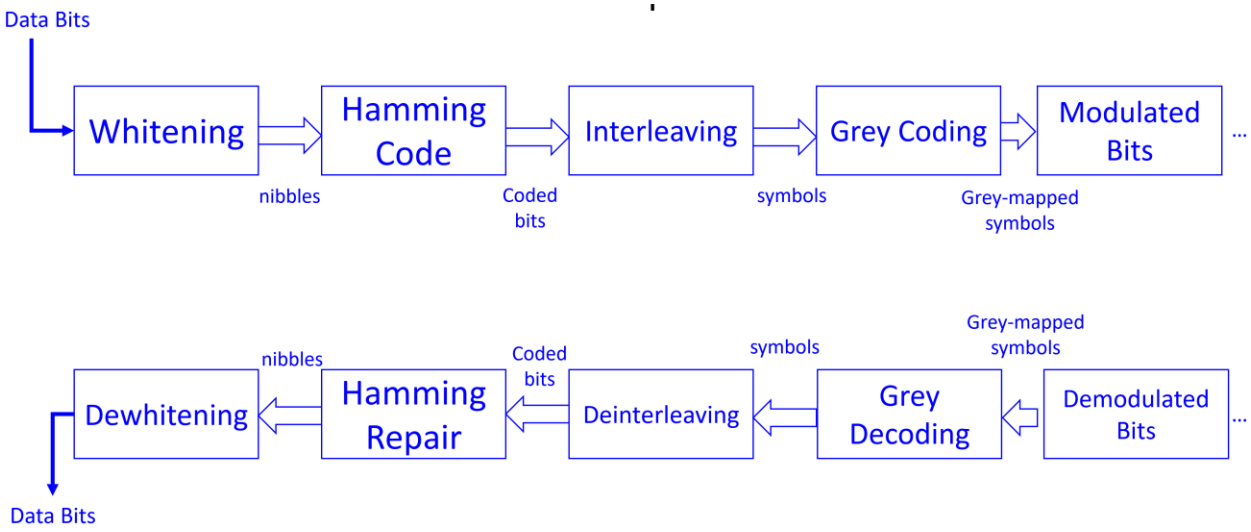
1.1. Structure



LoRa packet structure typically starts with a preamble of 8 upchirps followed by a SYNC symbol which contains two modulated upchirps (bin 8 and bin 16 in our case) followed by two downchirps and a quarter downchirp. The rest of the packet denotes the computed data modulated at the respective spreading factor.

The data is modulated as described in the paper below. However, broadly the modulated bits are computed based on the following steps:

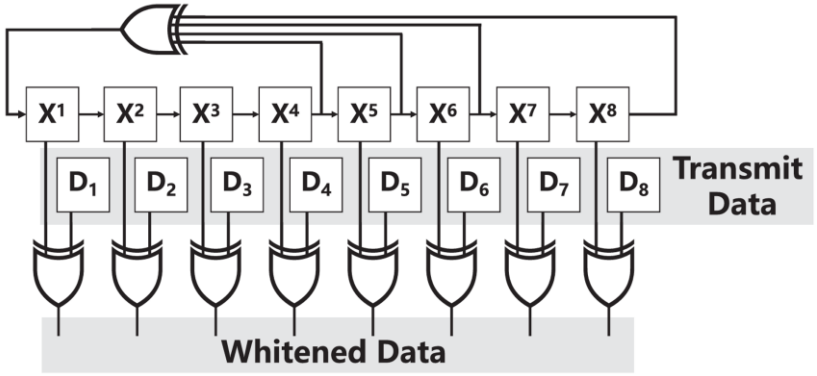
[1] This project is based on the description of the paper: https://dl.acm.org/doi/10.1145/3546869



1.1.1 Whitening

Whitening ensures that the data is randomized enough to avoid containing preamble symbols followed by the SYNC symbols in the middle of the packet.

LoRa uses the following Linear Feedback Shift Register (LFSR) to generate a pseudorandom sequence (generated sequence is provided to you)



$$x^8 + x^6 + x^5 + x^4 + 1$$

```
whitening_seq = uint8([0xff, 0xfe, 0xfc, 0xf8, 0xf0, 0xe1, 0xc2, 0x85, 0xb, 0x17,
0x2f, 0x5e, 0xbc, 0x78, 0xf1, 0xe3, 0xc6, 0x8d, 0x1a, 0x34, 0x68, 0xd0, 0xa0, 0x40,
0x80, 0x1, 0x2, 0x4, 0x8, 0x11, 0x23, 0x47, 0x8e, 0x1c, 0x38, 0x71, 0xe2, 0xc4, 0x89,
0x12, 0x25, 0x4b, 0x97, 0x2e, 0x5c, 0xb8, 0x70, 0xe0, 0xc0, 0x81, 0x3, 0x6, 0xc,
0x19, 0x32, 0x64, 0xc9, 0x92, 0x24, 0x49, 0x93, 0x26, 0x4d, 0x9b, 0x37, 0x6e, 0xdc,
0xb9, 0x72, 0xe4, 0xc8, 0x90, 0x20, 0x41, 0x82, 0x5, 0xa, 0x15, 0x2b, 0x56, 0xad,
0x5b, 0xb6, 0x6d, 0xda, 0xb5, 0x6b, 0xd6, 0xac, 0x59, 0xb2, 0x65, 0xcb, 0x96, 0x2c,
0x58, 0xb0, 0x61, 0xc3, 0x87, 0xf, 0x1f, 0x3e, 0x7d, 0xfb, 0xf6, 0xed, 0xdb, 0xb7,
0x6f, 0xde, 0xbd, 0x7a, 0xf5, 0xeb, 0xd7, 0xae, 0x5d, 0xba, 0x74, 0xe8, 0xd1, 0xa2,
0x44, 0x88, 0x10, 0x21, 0x43, 0x86, 0xd, 0x1b, 0x36, 0x6c, 0xd8, 0xb1, 0x63, 0xc7,
0x8f, 0x1e, 0x3c, 0x79, 0xf3, 0xe7, 0xce, 0x9c, 0x39, 0x73, 0xe6, 0xcc, 0x98, 0x31,
0x62, 0xc5, 0x8b, 0x16, 0x2d, 0x5a, 0xb4, 0x69, 0xd2, 0xa4, 0x48, 0x91, 0x22, 0x45,
0x8a, 0x14, 0x29, 0x52, 0xa5, 0x4a, 0x95, 0x2a, 0x54, 0xa9, 0x53, 0xa7, 0x4e, 0x9d,
0x3b, 0x77, 0xee, 0xdd, 0xbb, 0x76, 0xec, 0xd9, 0xb3, 0x67, 0xcf, 0x9e, 0x3d, 0x7b,
0xf7, 0xef, 0xdf, 0xbf, 0x7e, 0xfd, 0xfa, 0xf4, 0xe9, 0xd3, 0xa6, 0x4c, 0x99, 0x33,
0x66, 0xcd, 0x9a, 0x35, 0x6a, 0xd4, 0xa8, 0x51, 0xa3, 0x46, 0x8c, 0x18, 0x30, 0x60,
0xc1, 0x83, 0x7, 0xe, 0x1d, 0x3a, 0x75, 0xea, 0xd5, 0xaa, 0x55, 0xab, 0x57, 0xaf,
0x5f, 0xbe, 0x7c, 0xf9, 0xf2, 0xe5, 0xca, 0x94, 0x28, 0x50, 0xa1, 0x42, 0x84, 0x9,
0x13, 0x27, 0x4f, 0x9f, 0x3f, 0x7f]');
```

1.1.2 Hamming Code

Hamming code is one type of error correcting code that communicates additional parity bits that enable packet error detection and forward error correction at the receiver. Based on the coding rate, the parity bits are described as follows:

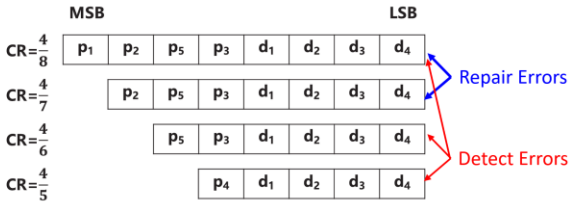
$p1 = d1 \oplus d2 \oplus d4$

$p2 = d1 \oplus d3 \oplus d4$

$p3 = d2 \oplus d3 \oplus d4$

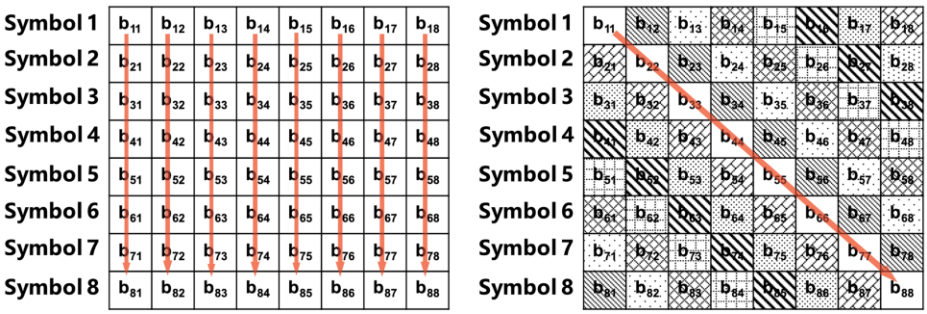
$p4 = d1 \oplus d2 \oplus d3 \oplus d4$

$p5 = d1 \oplus d2 \oplus d3$



1.1.3 Interleaving

LoRa uses interleaving to distribute symbol errors over the air across codewords to enable improved packet reception. Specifically, LoRa uses standard diagonal interleaving to mix the bits across symbols modulated over the air as described below:

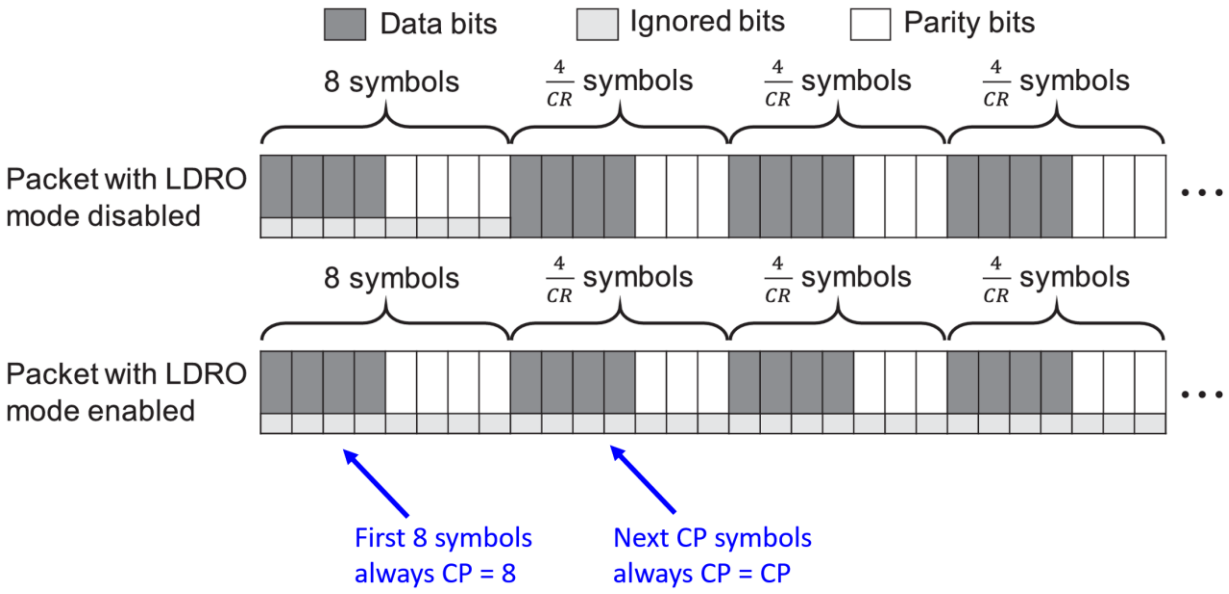


1.1.4 Grey Coding

Grey coding is used to map the distance between the physical data being sent and the hamming distance that our coding uses to secure data bits. Broadly, it simply maps k-bit codewords to other k-bit entities, one-on-one. Read the paper for more details.

1.1.5 LDRO Mode

For longer packets, the lower data bits of the are always kept 01 to protect them from center frequency offset (CFO) based errors. This reduces the data rate further for longer packets yet improves reception.



2. Project Goals

The goals of this project as the following:

- Demonstrate the ability to decode the LoRa protocol.
- Write a code to decode the LoRa packet in Python.
- Make the decoder resilient to various wireless impairments such as noise, center frequency offset.

3. How to Start?

Use the training data

1. You are being provided some data with known data bits and an encoder that generates outputs at every stage (use debugger to get intermediate stage outputs)
2. Start with the first step of grey decoding and try to match with the encoder output prior to grey encoding.
3. Keep continuing to reverse engineer the steps one by one. (Interleaving will probably be the toughest)
   [Training Data Payload is [1,2,3,4,80,73,78,71,8,9,10,11,12,13,14,15]]

Run the code on test data to verify operation

1. Check if your code works well with the test data.
2. See if the message output makes sense (it will be a reasonable word)

Test noise resilience of your code

1. Add noise to some of the sample data using commpy.channels.awgn[2]
2. Check the limit of the detection and decoding of the code you have written and improve it if possible.

4. Hand-in Procedure

You will submit your project as a python file to Canvas, and we will test your codes using an automated script. Create a file named README in the primary submission directory so we know that it is the primary directory that we need to grade (we will ignore the other).

5. Deliverable Items

The deliverables are enumerated as follows,

a. **The source code** for the entire project (if you use external libraries, provide the binaries/steps needed for compilation of your project)

[2] https://commpy.readthedocs.io/en/latest/generated/commpy.channels.awgn.html

b. A brief design document regarding your wireless decoder design in design.txt / design.pdf in the submission directory (1-2 pages). Tell us about the following,
    i. Your decoder design, model and components (add pictures if it helps)
    ii. How did you compensate for CFO? Is there a cool new way you found?
    iii. How does your code improve noise resilience?
    iv. How does your code deal with LDRO mode?
    v. Extra capabilities you implemented (optional, anything which was not specified)

6. Expected Submitted Code Operation

We will expect a file called LoRaDecoder.py in the submission which will operate as follows:
$ python LoRaDecoder.py <filename> <SF> <Payload length> <Coding Parameter>
where the file contains a single packet.
All files provided will be sampled at 250 ksps and have a bandwidth of 125 kHz.
None of the files will have a header but will contain a CRC.

The output should be as follows
<Packet start index [zero-indexed]> \n
<SF-bit demodulated symbols> \n
<Decoded message as uint8> \n

7. Grading

Partial credit will be available according to the following grading scheme,

Items	Points
Logistics - Design documents and code	**10** 10
Packet Detection and Decoding - Packet Start Detection - Packet Demodulation - Packet Decoding	**40** 10 10 20
Dealing with AWGN noise - Packet Start Detection at -15 dB SNR - Packet Decoding at -10 dB SNR - Packet Detection Limits (till -30 dB) - Packet Decoding Limits (till -20 dB)	**50 (+10)** 20 20 10 10