VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vývoj univerzálního analyzátoru pro LoRaWAN s využitím softwarově definovaného rádia

Development of Universal Analyzer for LoRaWAN Using Software-Defined Radio

Bc. Vladimír Dobeš

Diplomová práce

Vedoucí práce: Ing. Libor Michalek, Ph.D.

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/22/071185
vsboee22023538

Zadání diplomové práce

Student: **Bc. Vladimír Dobeš**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Vývoj univerzálního analyzátoru pro LoRaWAN s využitím softwarově definovaného rádia
Development of Universal Analyzer for LoRaWAN Using Software-Defined Radio

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vývoj a implementace analyzátoru komunikace s využitím LoRa modulace za použití softwarově definovaného rádia (SDR) ve frekvenční oblasti 868 MHz. Součástí řešení bude vývoj softwarového nástroje pro demodulaci, kanálové dekódování, zdrojové dekódování, dešifrování a zobrazení užitečných dat (payload) v LoRaWAN.

Zásady pro vypracování:
1. Popište technologii LoRaWAN.
2. Proveďte rešerši v oblasti dostupných řešení pro analýzu LoRaWAN.
3. Pomoci softwarově definovaného rádia (SDR) implementujte software, který bude schopen přijmout vyslaný rámec z koncentrátorů a koncových zařízení ve frekvenčním pásmu 867-870 MHz. Na přijatý rámec navrhněte algoritmy pro jeho dekódování, které budou funkční pro všechny činitele rozprostření a kódovací poměry.
4. Navrhněte a sestavte vizualizační prostředí v podobě tabulky a waterfall grafu obsahující úspěšně i neúspěšně zrekonstruovaná data. Waterfall graf by měl mít anotaci, která bude odkazovat do tabulky a naopak.
5. Výsledné řešení zhodnoťte.

Seznam doporučené odborné literatury:

[1] MARQUET, Alexandre; MONTAVONT, Nicolas; PAPADOPOULOS, Georgios Z. Towards an SDR implementation of LoRa: Reverse-engineering, demodulation strategies and assessment over Rayleigh channel. *Computer Communications* , 2020, 153: 595-605.
[2] ROBYNS, Pieter, et al. A Multi-Channel Software Decoder for the LoRa Modulation Scheme. In: *IoTBDS*. 2018. p. 41-51.

17. listopadu 2172/15
708 00 Ostrava-Poruba
Česká republika

spojovatelka: +420 597 321 111
epodatelna: epodatelna@vsb.cz
ID datové schránky: d3kj88v

IČ: 61989100
DIČ: CZ61989100

email: studijni.fei@vsb.cz
www.fei.vsb.cz

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Libor Michalek, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního oboru: doc. Ing. Petr Šiška, Ph.D.

V IS EDISON zadáno: 06.10.2022 10:34:44

Abstrakt

Práce se zabývá vývojem analyzátoru pro příjem LoRaWAN rámců ve frekvenční oblasti 868 MHz za použití softwarově definovaného rádia pro účely ladění a analýzy zachycené komunikace. Cílem práce je vyvinout software, který by byl schopen pro standardně používané konfigurace přijmout LoRaWAN rámce, demodulovat je a nakonec dekódovat, aniž by byla známá jediná informace o přijatém signálu. Za tímhle účelem byly vyvinuty metody pro detekci šířky pásma a činitele rozprostření, které dosahují vysoké úspěšnosti. Pro zrekonstruování dat byl implementován demodulátor a dekódér, kdy výstup je zprostředkován do softwaru Wireshark. V práci je hojně využíváno pokročilých prostředků pro digitální zpracování signálu jako například FFT a STFT, pracuje se též s analytickým signálem reprezentujícím průběh okamžitého kmitočtu přijatého signálu. Implementovaný software je podroben testu o několika krocích, kde je ověřena jeho funkčnost. V závěru je celé řešení zhodnoceno včetně jeho nedostatků, čímž se otevírají možné směry pro budoucí zkoumání.

Klíčová slova

LoRa, LoRaWAN, SDR, Analyzátor, DSP, Python, Adalm-Pluto SDR, PlutoSDR

Abstract

The thesis deals with the development of an analyzer for the reception of LoRaWAN frames in the 868 MHz frequency band using a software defined radio for the purposes of debugging and analysis of captured communication. The goal of the thesis is to develop software that would be able to receive LoRaWAN frames for standard configurations, demodulate them and finally decode them without knowing any preliminary information about received signal. For this purpose, methods for Bandwidth determination and Spreading Factor determination have been developed. Both methods achieved high success rates. To reconstruct the data, a demodulator and a decoder were implemented. The output of the decoder is passed into the Wireshark software. Advanced tools for digital signal processing such as FFT and STFT are extensively used in the work and an analytical signal representing the instantaneous frequency of the received signal is also used during signal processing. The implemented software is subjected to a test of several steps, where its functionality is verified. In the conclusion, the entire solution is evaluated, including its drawbacks, opening up possible directions for future research.

Keywords

LoRa, LoRaWAN, SDR, Analyzer, DSP, Python, Adalm-Pluto SDR, PlutoSDR

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Liborovi Michalkovi, Phd. a Ing. Marku Novákovi z ACRIOS Systems s.r.o. za odbornou pomoc, vstřícnost při konzultacích a za rady při zpracování této práce. Také bych rád poděkoval Ing. Janu Královi, Phd. a Ing. Ondřeji Kolářovi z UREL FEKT VUT za poskytnutí souboru pro zprostředkování dekódovaných dat do softwaru Wireshark. Rovněž bych rád poděkoval společnosti ACRIOS Systems s.r.o za propůjčení LoRa a LoRaWAN modulů ACR-CV pro otestování funkčnosti analyzátoru.

Obsah

Seznam použitých zkratek a symbolů

SDR	–	Software Defined Radio
DSP	–	Digital Signal Processing
CSS	–	Chirp Spread Spectrum
IoT	–	Internet of Thing
BW	–	Bandwidth
SF	–	Spreading Factor
CR	–	Coding Rate
FEC	–	Forward Error Correction
IP	–	Internet Protocol
RSSI	–	Received Signal Strength Indicator
AES	–	Advanced Encryption Standard
OTAA	–	Over-The-Air Activation
ABP	–	Activation By Personalization
MIC	–	Message Integrity Code
MAC	–	Medium Access Control
CRC	–	Cyclic Redundancy Check
ETSI	–	European Telecommunications Standards Institute
SNR	–	Signal to Noise Ratio
USB	–	Universal Serial Bus
FFT	–	Fast Fourier transform
FIR	–	Finite Impulse Response
IIR	–	Infinite Impulse Response
XOR	–	eXclusive OR
LSB	–	Least Significant Bit
LFSR	–	Linear-Feedback Shift Register

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Cílem práce je pomocí softwarově definovaného rádia (SDR) implementovat software, který bude schopen přijmout vyslaný rámec z koncentrátorů a koncových zařízení ve frekvenčním pásmu 867-870 MHz, demodulovat a dekódovat jej a zrekonstruovaná data zprostředkovat. Motivací je mít nástroj, který lze použít pro příjem LoRa/LoRaWAN zařízení v reálném provozu, díky čemuž je lze snadněji analyzovat a ladit. Zároveň by u takového nástroje měl být kladen důraz zvláště na přenositelnost, výpočetní náročnost pro možnost zachytit co největší části komunikace a být kompatibilní se standardně používanými rozhraními.

Obsah práce kopíruje zadání k diplomové práci. Nejdřív je stručně popsána modulace LoRa, jelikož úzce souvisí s technologii LoRaWAN. Důraz je zde především kladen na pojem činitel rozprostření, protože se s ním lze potkat v celé práci. Následně jsou uvedeny hlavní informace k technologii LoRaWAN jako architektura sítě, či frekvenční plán.

Poté je provedena rešerše vybraných dostupných nástrojů pro analýzu technologie LoRaWAN, respektive pro modulaci LoRa, kde je uvedeno celkově pět řešení. Zde jsou popsány jejich možnosti, nedostatky, případně i osobní zkušenost s nimi.

Po rešerši následuje implementace navrženého softwaru. Je zde popsáno jaké prostředky byly použity, a proč byly vybrány pro implementaci analyzátoru. Jelikož se software větví do několika souborů, je popsána jeho souborová struktura a uvedeno blokové schména analyzátoru. Následně je popsán účel jednotlivých bloků a nástrojů uvnitř nich, jak fungují, popřípadě co bylo vyzkoušeno, ale nebylo použito.

V rámci práce je také zprostředkováno testování, kde je blíže popsáno, jak byla ověřena funkčnost analyzátoru včetně uvedení výstupních dat z testování v příloze na konci práce.

V závěru je celé řešení zhodnoceno, shrnuto, zda byly splněny cíle práce a její přínosy. Rovněž byly uvedeny nedostatky analyzátoru, které otevírají možnost pokračovat v práci formou například bakalářské práce, a nebo jiné formy výzkumu.

Kapitola 2

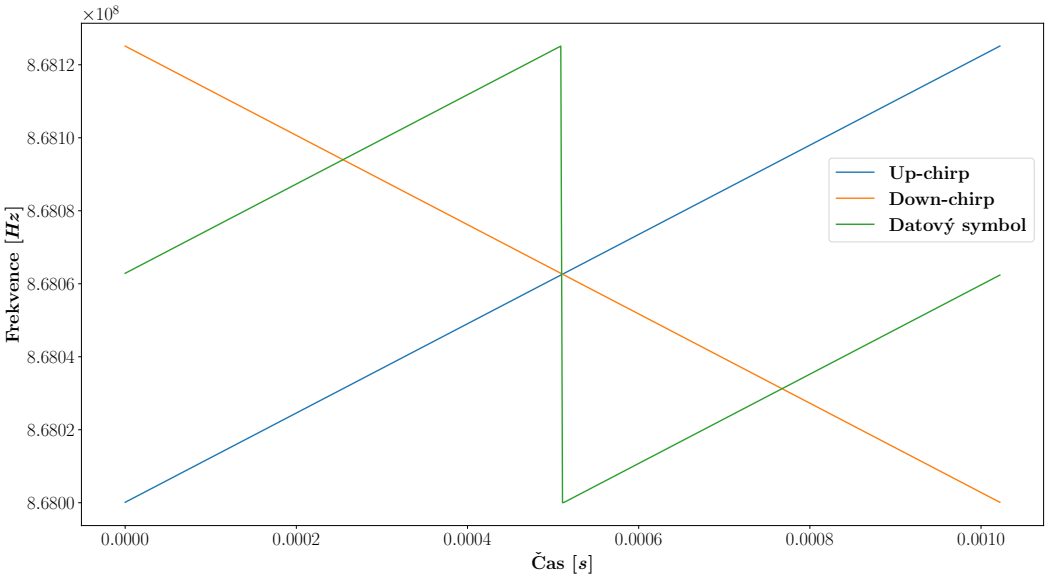
Modulace LoRa

LoRa je bezdrátová technologie na fyzické vrstvě, která je odvozena od CSS modulace. Díky malé energetické náročnosti a velkému komunikačnímu dosahu našla využití především v IoT zařízeních jako jsou senzory, nebo aktuátory. Dosah v zastavěné oblasti je až 5 kilometrů, mimo ní až 15 kilometrů. Zařízení používající LoRa technologii často komunikují jen párkrát denně, zprávy bývají zpravidla malé a zbytek času, co zařízení nekomunikuje je ve spánku, kdy odběr býva v mili wattech. Když se uvedené aspekty sečtou společně s nízkou energetickou náročnosti modulace, dokáže zařízení být napájeno z baterie až 10 let. [1] [2]

2.1 Princip modulace

LoRa modulace spadá do rodiny spektra rozprostírajících modulací. V případě LoRa modulace je rozprostření spektra dosaženo generováním chirp signálu, jehož frekvence se v průběhu času zvyšuje, a nebo snižuje, kde jeden chirp signál představuje jeden symbol. Obecně lze chirp signály dělit na 2 základní typy: na up-chirp a na down-chirp. V případě prvního uvedeného typu se frekvence zvyšuje z nejnižší možné na nejvyšší možnou. V druhém případě je to přesně naopak - frekvence se snižuje z nejvyšší možné na nejnižší možnou. Odkud kam bude daný chirp přeladovat, určuje šířka pásma (BW). [3] [4]

Jinak řečeno, pokud bychom uvažovali, že používáme up-chirp signál, který má počateční frekvenci 868,000 MHz a šířka pásma je 125 kHz, jeho frekvence by se postupně zvyšovala až do koncové frekvence, která je v tomhle případě 868,125 MHz. U uplink zpráv se pro datové symboly využívají převážně up-chirp signály s tím rozdílem, že počáteční frekvence se odvíjí od modulované hodnoty. Pokud by tedy byla modulována nějaká hodnota, u které vychází, že pro daný chirp je počáteční frekvence 868,0625 MHz, kde šířka pásma je opět 125 kHz a kanál je na 868,0000 MHz. Tak v takovém případě chirp signál kmitá do frekvence 868,1250 MHz, poté se přeladí na frekvenci 868,0000 MHz a kmitá až do frekvence 868,0625 MHz.



Obrázek 2.1: Ukázka jednotlivých chirp symbolů

2.2 Spreading Factor

Činitel rozprostření (Spreading Factor) vyjadřuje, kolika bity je symbol vyjádřen. Obecně se používá 6 různých činitelů rozprostření: SF7-SF12. Symbol tedy může obsahovat celkově 2^{SF} různých hodnot (nebo také tzv. čipy). Použije-li se například SF7, znamená to, že symbol je tvořen sedmi bity, a tudíž může obsahovat celkově $2^7 = 128$ hodnot. [3] [4]

`Chip Rate`, nebo-li počet přenesených/přijatých čipů za sekundu se dá vyjádřit jako:

$$R_c = BW \tag{2.1}$$

Zná-li se vztah pro výpočet čipů na symbol i R_c lze vyjádřit `Symbol Rate` vztahem:

$$R_s = \frac{R_c}{2^{SF}} \tag{2.2}$$

Dobu trvání jednoho čipu lze vyjádřit následovně:

$$T_c = \frac{1}{R_c} = \frac{1}{BW} \tag{2.3}$$

A podobně i dobu trvání jednoho symbolu:

$$T_s = \frac{1}{R_s} = \frac{2^{SF}}{BW} \tag{2.4}$$

Jelikož činitel rozprostření udává, kolik bitů je použito v jednom symbolu, lze vyjádřit `Bit Rate` pomocí činitele rozprostření a symbolové rychlosti vztahem:

$$R_b = SF \cdot R_s \tag{2.5}$$

Protože jsou jednotlivé LoRa symboly opatřeny dopřednou korekci chyb (FEC), je skutečná bitová rychlost vypočítána jako:

$$R_b = SF \cdot R_s \cdot CR \tag{2.6}$$

kde CR je kódovací poměr a může nabývat celkově 4 hodnot - $\{\frac{4}{5}, \frac{4}{6}, \frac{4}{7}, \frac{4}{8}\}$. [4]

2.2.1 Vlastnosti činitele rozprostření

Činitele rozprostření jsou ortogonální. Signály na stejné frekvenci s odlišným činitelem rozprostření spolu nekolidují - jeden druhému se jeví jako šum. Naopak signály se stejným činitelem rozprostření můžou zkolidovat. Výjimkou, kdy signál může být úspěšně zpracován, je, pokud je signál oproti druhému silnější o 6 dB. [1] [5]

Přenosová rychlost se mění v závislosti na použitém činiteli rozprostření - pokud se činitel rozprostření zvyšuje, přenosová rychlost se zvyšuje. Obdobně je to se šířkou pásma. Za předpokladu, že činitel rozprostření i kódovací poměr je stejný, a je-li šířka pásma zdvojnásobena, je i přenosová rychlost dvakrát vyšší.

Naopak se zvyšujícím se činitelem rozprostření stoupá procesní zisk (`processing gain`). Signály tak můžou být přijaty s menším poměrem chyb, a proto se zvyšujícím se činitelem rozprostření stoupá i vzdálenost, kterou signál dokáže urazit. Vyšší činitelé rozprostření mají také oproti menším vyšší dobu vysílání (`Time-on-Air`) a vyšší přijímací citlivost (viz. Tabulka 2.1). [5]

Spreading Factor	**Citlivost přijímače [dBm]**
SF7	-123,0
SF8	-126,0
SF9	-129,0
SF10	-132,0
SF11	-134,5
SF12	-137,0

Tabulka 2.1: Citlivost přijímače pro jednotlivé činitele rozprostření při šířce pásma 125 kHz [5]

2.3 Dopředná korekce chyb (FEC)

Dopředná korekce chyb je technika, která na základě redundantních bitů posílaných společně s datovými bity, dokáže na straně přijímače detekovat, případně opravit chyby, které mohly nastat během přenosu. Přijímač při příjmu vykoná potřebné úkony a pokud nalezne chybu, může ji (pokud

je to možné na základě použité techniky) opravit. Dopřednou korekci chyb lze obecně dělit na dvě skupiny - blokové kódy a konvoluční kódy. [6]

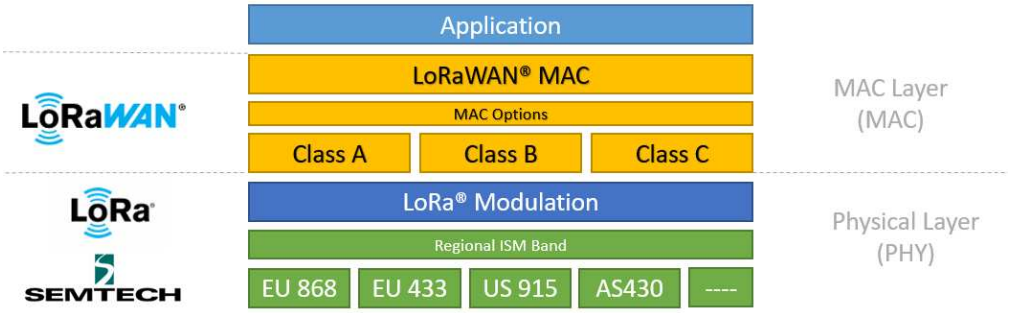
Pro technologii LoRa se používá sudá parita pro CR 4/5 a pro ostatní kódovací poměry jsou použity (k, n) Hammingovy kódy, kdy platí, že $k = 4$ a $n \in \{6, 7, 8\}$, kde k jsou informační prvky a n je velikost kódového slova. (4, 6) Hammingův kód je punkturovanou verzí (4, 7) Hammingova kódu, zatímco (4, 8) Hammingův kód je rozšířenou verzí (4, 7) Hammingova kódu. Při použití CR 4/7 a CR 4/8 lze opravit jednu bitovou chybu. [7] [8] [9]

U Hammingových kódů jsou pro vytvoření kódových slov, nebo jejich korekci využívány generující a kontrolní matice. *"Na vysílací straně, získáme vysílací kódové slovo B vynásobením zprávy M generující maticí G ... Na přijímací straně vynásobíme kontrolní matici H přijatým kódovým slovem B' a získáme syndrom S ... Pokud syndrom $S = 0$, tak chyba nanastala, nebo při přenosu mohlo vzniknout více chyb. Pokud je syndrom $S \neq 0$, tak tento vektor nám udává pozici, kde nastala chyba".* [9]

Kapitola 3

LoRaWAN

LoRaWAN je otevřený síťový protokol standardizovaný a udržovaný asociací LoRa Alliance, který zajišťuje obousměrnou komunikaci, mobilitu a lokalizační služby. Pro vytvoření komunikačního kanálu na fyzické vrstvě je využívána LoRa. [1]



Obrázek 3.1: LoRaWAN stack [1]

3.1 Frekvenční plán

V Evropě lze provozovat zařízení podporující LoRaWAN v pásmech EU863-870 a EU433. Pro obě pásma platí, že zařízení by měla podporovat tři výchozí kanály a měla by podporovat 6 konfigurací pro rychlost přenosu dat (DR0-DR5), viz. Tabulka 3.1. Výchozí kanály by měly být na kmitočtech 868,10 MHz, 868,30 MHz 868,50 MHz pro pásmo EU863-870, respektive 433,175 MHz, 433,375 MHz, 433,575 MHz pro pásmo EU433. [10]

3.2 Architektura sítě

LoRaWAN síť je primárně hvězdicové topologie.

DR	Konfigurace	Přenosová rychlost [bit/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF12 / 250 kHz	11000
7	FSK: 50 kbps	50000
8..14	RFU	
15	LinkADRReq MAC příkaz	

Tabulka 3.1: Přenosové rychlosti a konfigurace [10]

3.2.1 Koncové zařízení

Je senzor, nebo aktuátor bezdrátově připojený do LoRaWAN sítě skrze koncentrátor, se kterým je ke komunikaci použita modulace LoRa. Ve většině případů je koncové zařízení autonomní, často baterii napájený senzor, který snímá fyzikální podmínky a enviromentální události. Koncové zařízení jako aktuátor se typicky používá pro pouliční osvětlení, bezdrátové zámky či pro úzavěr vodního ventilu.

3.2.2 Koncentrátor

Koncentrátor operuje zcela jen na fyzické vrstvě, kde přeposílá jednotlivé LoRa zprávy mezi koncovým zařízením a síťovým server. Přijímá LoRa modulované zprávy od kteréhokoliv zařízení v jeho dosahu a pokud je zachována integrita dat, přeposílá jednotlivé zprávy společně s metadaty, RSSI a časovým razítkem skrz IP spojení do síťového serveru. Pokud je v dosahu zařízení více koncentrátorů, příjmou zprávu všichni. Následnou duplicitu řeší síťový server a na základě RSSI hodnot si volí nejvhodnější koncentrátor pro komunikaci s daným zařízením.

3.2.3 Síťový server

Síťový server má na starost správu celé sítě - dynamicky kontroluje parametry sítě k adaptování systému na měnící se podmínky, vytváří zabezpečené 128 bitové spojení pro transport dat z koncového zařízení do aplikačního serveru, kontroluje přicházející provoz z koncových zařízení a zpátky. Také zajišťuje autenticitu každého koncového zařízení v síti a integritu každé zprávy. Nicméně i přes to, že skrz něj prochází všechny zprávy, nemá přístup k datům aplikace.

3.2.4 Aplikační server

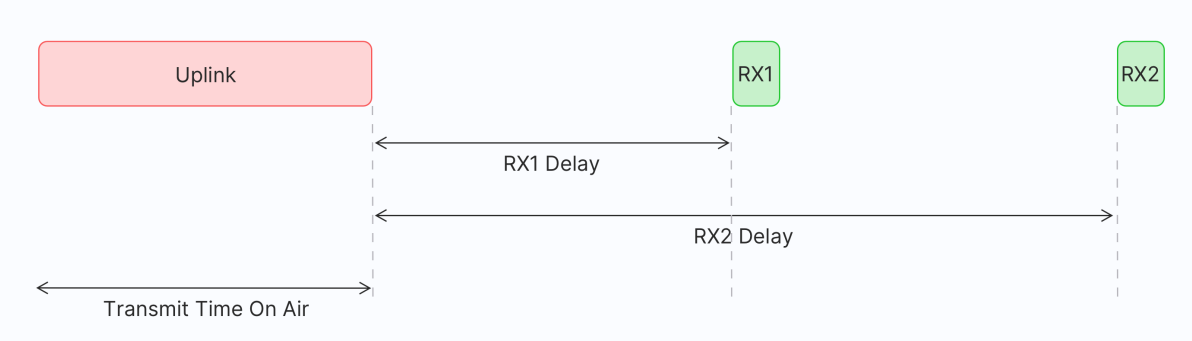
Aplikační server je zodpovědný za bezpečné zpracování, správu a interpretaci dat z koncového zařízení. Stojí také za generováním všech dat na aplikační vrstvě pro připojená koncová zařízení. [1]

3.3 Třídy zařízení

Pro rozmanitost aplikací jsou v rámci LoRaWAN definované tři třídy zařízení - A, B a C.

3.3.1 Třída A

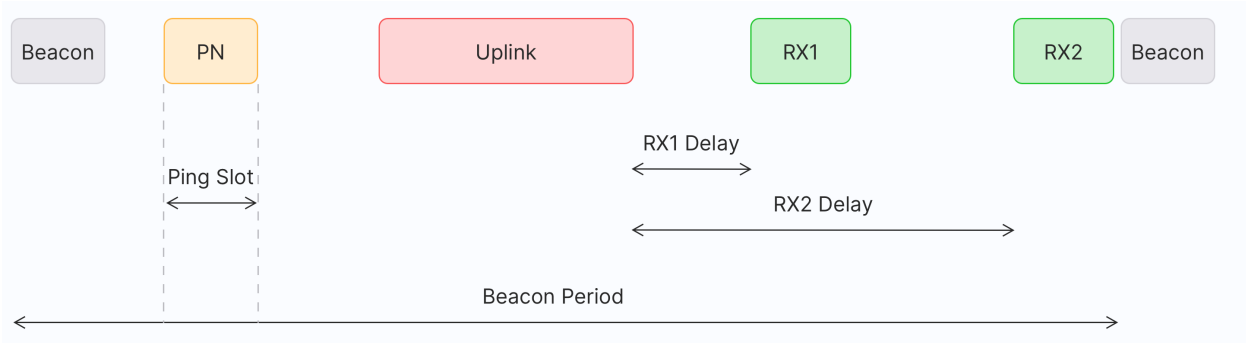
Je výchozí třída, kterou musí podporovat všechny LoRaWAN koncová zařízení. Komunikace je vždy zahájena koncovým zařízením, kdy jakmile pošle zprávu směrem do sítě, otevře s malým zpožděním dvě krátká okna, ve kterých přijímá případné zprávy ze sítě. Pokud žádná zpráva nedorazí, musí síťový server počkat na další zprávu ze zařízení. Díky tomu může zařízení přejít na dobu určenou aplikací do nízkovýkonného spícího režimu, což dělá třídu A ze všech tří nejméně energeticky náročnou.



Obrázek 3.2: Třída A přijímací okna [11]

3.3.2 Třída B

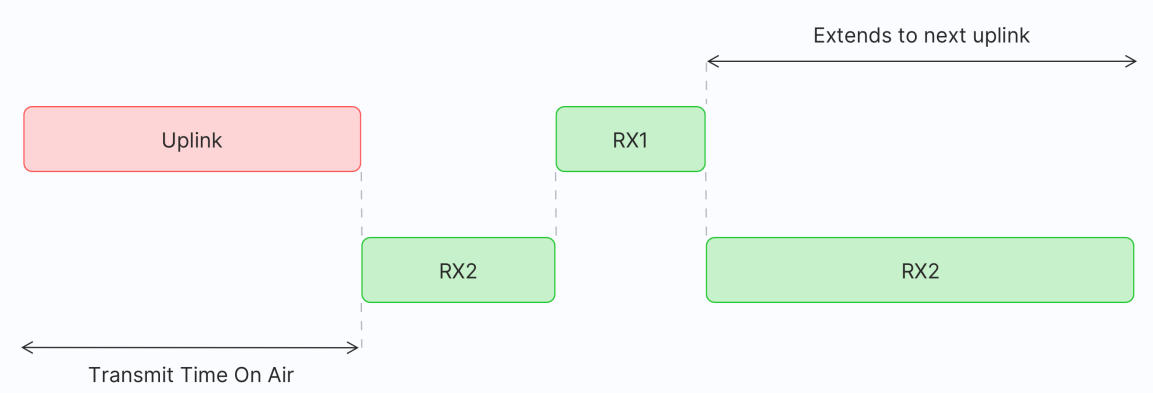
Třída B vychází z třídy A s tím rozdílem, že koncové zařízení otevírá ještě jedno naplánované přijímací okno - tzv. ping slot. To se otevírá na základě časově synchronizované beacon zprávy vysílané koncentrátorem. Koncová zařízení třídy B oproti koncovým zařízením třídy A poskytují menší latenci, protože jsou k zastižení v předdefinovaných časech. Na druhou stranu, kvůli dalšímu přijímacímu oknu a beacon zprávám vydrží koncová zařízení třídy B oproti koncovým zařízením třídy A kratší dobu na napájení z baterie.



Obrázek 3.3: Třída B přijímací okna [11]

3.3.3 Třída C

Třída C rozšiřuje třídu A. Koncová zařízení kromě doby kdy vysílají, nechávají přijímací okna neustále otevřená. Díky tomu je zde nulová latence, protože síťový server může kdykoliv iniciovat komunikaci směrem ke koncovému zařízení. Daň za to je vyšší spotřeba než v předchozích dvou případech, a proto je pro zařízení operující v třídě C přivětivější trvalé napájení z elektrické sítě. [11] [12]



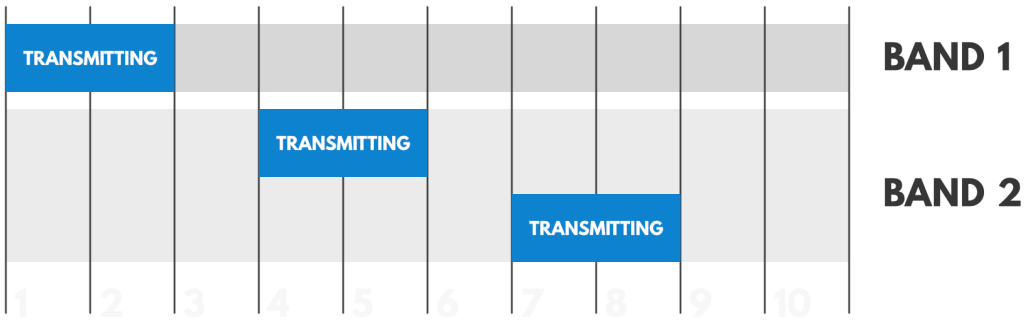
Obrázek 3.4: Třída C přijímací okna [11]

3.4 Klíčovací poměr

Klíčovací poměr (Duty cycle) je regulace vydaná organizací ETSI, která přináší restrikce v podobě, jak dlouho vysílač může být zapnutý, nebo také maximální dobu, kterou může vysílač vysílat po dobu jedné hodiny [10]. Restrikce se vztahují na koncová zařízení i na koncentrátory [13].

Znamená to tedy, že pokud zařízení vysílá na kanálu po dobu dvou časových jednotek z celkových deseti časových jednotek, je klíčovací poměr roven 20 %. Vysílá-li zařízení na třech kanálech při zachování totožných parametrů činí klíčovací poměr 60 %. Situace se trochu mění, pokud jsou

použity kanály v jiných podpásmech (viz obrázek 3.5), což je běznou praxí v evropském frekvenčním plánu. Například pokud by byly zachovány tři kanály z předchozího případu se stejnými parametry, kdy jeden kanál je v jednom podpásmu a zbylé dva v druhém podpásmu, klíčovací poměr pro první podpásmo by činil 20 %, zatímco v druhém případě by byl roven 40 %. I přesto, že celkový klíčovací poměr daného zařízení je 60 %, byl celkový vysílací čas rozložen do dvou podpásem, čímž lze zvýšit celkovou vysílací dobu, aniž by byly porušeny regulace vydané organizací ETSI. [13]



Obrázek 3.5: Příklad klíčovacího poměru pro dvě podpásma [13]

3.5 Zabezpečení

Dle LoRaWAN 1.0 specifikace se využívají 128 bitové AES klíče.

3.5.1 Relační klíče

Při připojení koncového zařízení do sítě je vygenerovaný aplikační relační klíč (`AppSKey`) a síťový relační klíč (`NwkSKey`). Oba klíče jsou unikátní pro každé koncové zařízení a jsou používány po dobu trvání relace. V případě použití metody OTAA jsou oba klíče opět generovány při každé nové aktivaci, v případě metody ABP zůstavají stejné do doby, dokud nejsou změněny.

- **Aplikační relační klíč** slouží pro šifrování a dešifrování užitečných dat. Ty jsou plně šifrována mezi koncovým zařízením a aplikačním serverem - to znamená, že nikdo kromě uživatele nemá přístup k příchozímu, nebo odchozímu obsahu zpráv.
- **Síťový relační klíč** se používá mezi koncovým zařízením a síťovým serverem. Za pomoci kódu integrity zprávy (`MIC`) je s ním kontrolována integrita každé zprávy.

3.5.2 Aplikační klíč

Aplikační klíč zná pouze samotné koncové zařízení a aplikace. Slouží k odvození relačních klíčů během aktivace v případě užití metody OTAA.

3.5.3 Kontrola rámců

Při odchycení zpráv nejde díky relačním klíčům s nimi manipulovat a ani je přečíst, ale je možné zachycenou zprávu znovu přeposlat. Proto jsou v protokolu LoRaWAN definované dva čítače (`FCntUp` a `FCntDown`). Jakmile je zařízení aktivováno, jsou oba čítače nastaveny na nulu. Při odeslání zprávy jsou oba čítače inkrementovány - `FCntUp` v případě koncového zařízení, `FCntDown` v případě sítě. Pokud koncovému zařízení/síti dorazí zpráva s hodnotou čítače menší, než byla v předchozím případě, zpráva je ignorována. [11]

3.6 Formát rámce

3.6.1 Fyzická vrstva

Formát fyzické vrstvy využívající LoRa modulaci je zobrazen v tabulce 3.2. Zprávy začínají preambulí. V případě použití explicitní hlavičky je preambule následována polem `PHDR`, které nese informace o délce pole `PHYPayload`, jaký kódovací poměr je použit a pokud je CRC součástí zprávy. `PHDR_CRC` je CRC hlavičky. Pokud je použita implicitní hlavička pole `PHDR` a `PHDR_CRC` nejsou přítomná. Následuje `L` bytů dat ve fyzické vrstvě. Integrita dat je chráněna 2 byty CRC (pokud je přítomno). [10]

Velikost	8 symbolů	8 symbolů		L bytů	2 byty
Struktura paketu	Preambule	PHDR	PHDR_CRC	PHYPayload	CRC (pouze při uplinku)

Tabulka 3.2: Formát paketu fyzické vrstvy [10]

3.6.2 Data na fyzické vrstvě

Všechny LoRaWAN pakety se skládají z MAC hlavičky (`MHDR`), užitečných dat MAC vrstvy (`MACPayload`) a končící kodem zachovající integritu zprávy (`MIC`). Na základě použitém frekvenčním plánu a použité rychlosti přenosu (`DR`) se odvíjí maximální délka dat (`M`) v poli `MACPayload`. Obecně platí, že s vyšším činitelem rozprostření je maximální délka `M` nižší. [10] [14]

Velikost (v oktetech)	1	7..M	4
PHYPayload	MHDR	MACPayload	MIC

Tabulka 3.3: Formát dat na fyzické vrstvě [14]

3.6.3 MAC vrstva

3.6.3.1 MAC hlavička (MHDR)

MAC hlavička obsahuje typ rámce (`FType`). Celkově je definováno 8 různých typu rámce - pro OTAA, nepotvrzované a potvrzované uplink/dowlink zprávy, jeden bit je vyhrazený pro budoucí užití a jeden pro propietární účely. Dva bity v MAC hlavičce jsou vyhrazeny pro pole `Major`. Pole `Major` určuje podobu prvních čtyř oktetů v MAC payloadu a jak probíhá výměna rámců při připojování zařízení do sítě. Tři bity jsou rezervované pro budoucí použití (`RFU`).

Bity	[7:5]	[4:2]	[1:0]
MHDR	FType	RFU	Major

Tabulka 3.4: Formát MAC hlavičky [14]

3.6.4 Data na MAC vrstvě

Pole `MACPayload` se dělí na hlavičku rámce (`FHDR`), dále nepovinným polem specifikující port (`FPort`) a nepovinnou části obsahující užitečná data (`FRMPayload`).

3.6.4.1 Hlavička rámce (FHDR)

V hlavičce rámce lze nalézt adresu koncového zařízení (`DevAddr`), pole s doplňkovými informacemi (`FCtrl`) a dále čítač `FCnt` (viz. kapitola 3.5.3). Hlavičku ukončuje pole `FOpts` pro možný přenos MAC příkazů. [14]

Velikost (v oktetech)	4	1	2	0..15
FHDR	DevAddr	FCtrl	FCnt	FOpts

Tabulka 3.5: Formát hlavičky rámce [14]

Kapitola 4

Implementace analyzátoru

4.1 Rešerše dostupných řešení pro analýzu LoRaWAN

Mezi jedno z nejstarších řešení patří implementace dostupná na [15]. První příspěvek do nástoje Github byl představen v létě 2016, zatímco poslední je datován na podzim roku 2017. Projekt řeší pouze přijímací část využitím bloků GNU rádia a softwarově definovaného rádia. S řešením lze předem definovat činitel rozprostření přijatého signálu, přijat jej a úspěšně zrekonstruovat data pro všechny činitele rozprostření i kódovací poměry. Řešení také podporuje implicitní hlavičky, downlink zprávy a redukovaný přenos dat. V rámci řešení bylo vyzkoušeno několik vysílačů a přijímačů, které vedly k úspěšnému zrekonstruování přijatých LoRa zpráv.

V dalším řešení (dostupného na [16]) je implementována vysílací i přijímací část opět pomocí GNU rádia. Autoři článku mají k dispozici jak github projekt, tak i video dostupné na video platofmě youtube, ve kterém lze pochopit jednotlivé kroky pro zrekonstruování přijatých dat. Při definování činitele rozprostření, šířky pásma a kódovacího poměru lze v rámci řešení úspěšně demodulovat a dekódovat LoRa zprávy s implicitní hlavičkou. Díky široké paletě literatury uvedeného řešení bylo z něj především čerpáno pro první pokus pro zprovoznění dekódéru. Nicméně implementovaný dekódér se zdál nefunkční a byly problémy s jeho zprovozněním, a proto bylo od uvedeného řešení upuštěno.

Obdobné řešení je dostupné na [17], které taky implementuje vysílací a přijímací část, ale pro projekt je použit software Pothos. Navíc by mělo být schopno dekódovat i explicitní hlavičku. Součástí řešení je také zprostředkovací část s různými vizuálními výstupy, z kterých si lze vzít inspiraci pro případný vývoj grafického rozhraní.

Další řešení pro GNU rádio je dostupné na [18], kde jako softwarově definované rádio bylo použito LimeSDR. Řešení především čerpá z implementace dostupné na [15]. V rámci řešení byla implementovaná vysílací i přijímací část. Ve vysílací části nastaly problémy s LoRa zprávami s CR=2 a SF={6,11,12}. Na přijímací části byl zaznamenán spolehlivý přenos pro SF={7, 8, 9, 10}.

Poslední uvedené řešení pro analýzu LoRa zpráv je dostupné na [7]. Projekt je stále udržován a dostává se mu poměrně častých aktualizací. Jako v ostatních řešeních, i zde je implementována vysílací i přijímací část pomocí bloků GNU rádia. Lze využít všechny činitele rozprostření (i 5 a 6), kódovací poměry, lze dekódovat implicitní i explicitní hlavičku a rozeznat, jestli je použit redukovaný přenos dat. Je možné využít i tzv. měkký dekódovací proces pro snížení výpočetní náročnosti. Z mého pohledu se jedná o nejsofistikovanější řešení ze všech uvedených, a proto je i z něj především čerpáno při implementaci dekódéru. Řešení bylo také poměrně jednoduché na zprovoznění, a tudíž bylo hojně využíváno pro ladění mnou vytvořené implementace.

4.2 Použité prostředky

Pro implementaci analyzátoru byl vybrán programovací jazyk Python. Především z důvodu, že s ním mám největší zkušenosti, ale taky kvůli oblibě a velkému množství knihoven, které by se případně daly použít pro zhotovení analyzátoru. I přesto, že se jedná o relativně málo výkonný programovací jazyk, tak to na běh programu nemá podstatný vliv. Většina operací napsaných v jazyce Python nemá vysokou výpočetní náročnost, a pokud ano jsou pro to využity externí knihovny napsané především v jazyce C, kde jazyj Python slouží jako rozhraní mezi oběma jazyky.

Obrázek 4.1: Adalm-Pluto SDR [19]

Jako softwarově definované rádio bylo použito Adalm-Pluto SDR. Především z důvodu, že se jednalo o zařízení, které jsem již měl před implementací k dispozici, ale zároveň zařízení splňuje všechny požadavky pro zhotovení LoRaWAN analyzátoru. Je lehce přenositelné i cenově dostupné. Dokáže operovat na kmitočtech od 325 MHz do 3,8 GHz, což pokrývá nejvíce využíváné frekvenční plány v Evropě pro provoz technologie LoRaWAN (EU433, EU863-870). Z uvedených frekvenčních plánu vyplývá, že teoreticky by stačilo pokrývat šířku pásma o velikosti 7 MHz. Nicméně Adalm-Pluto SDR dokáže využít šířku pásma až 20 MHz, tudíž je tenhle parametr až nadstandardní pro analýzu

LoRaWAN provozu. Je vybaven napájecím rozhraním USB 2.0 s Micro-USB 2.0 kontektorem, které jsou v dnešní době velmi dostupné a populární. Také obsahuje jeden přijímač i jeden vysílač, kdy lze využít jak half-duplex, tak i full-duplex. Existuje tedy možnost implementovat i vysílací část, která by mohla přijít vhod při různém ladění reálné LoRaWAN instalace. A v neposlední řádě obsahuje Python API. [19]

4.2.1 Zprovoznění Adalm-Pluto SDR pro programovací jazyk Python

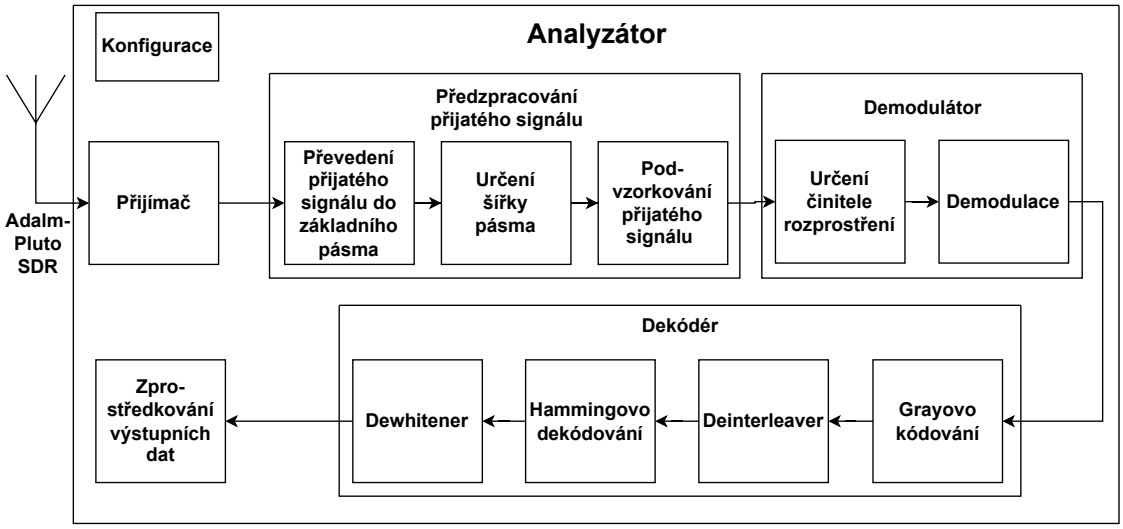
Níže uvedený postup (dostupný na [20]) byl použit k nainstalování všech potřebných balíčků pro práci s Adalm-Pluto SDR v programovacím jazyce Python. Jako operační systém byl využíván Linux Ubuntu 21.10 a Linux Ubuntu 22.04.

```
sudo apt-get install build-essential git libxml2-dev bison flex libcdk5-dev cmake
    python3-pip libusb-1.0-0-dev libavahi-client-dev libavahi-common-dev libaio-
    dev
cd ~
git clone --branch v0.23 https://github.com/analogdevicesinc/libiio.git
cd libiio
mkdir build
cd build
cmake -DPYTHON_BINDINGS=ON ..
make -j$(nproc)
sudo make install
sudo ldconfig

cd ~
git clone https://github.com/analogdevicesinc/libad9361-iio.git
cd libad9361-iio
mkdir build
cd build
cmake ..
make -j$(nproc)
sudo make install

cd ~
git clone https://github.com/analogdevicesinc/pyadi-iio.git
cd pyadi-iio
pip3 install --upgrade pip
pip3 install -r requirements.txt
```

```
sudo python3 setup.py install
```

4.3 Souborová struktura analyzátoru

Implementace analyzátoru je pro přehlednost logicky rozdělena do sedmi souborů, každý vykonávající svou specifickou roli. Spolu pak tvoří celek, s kterým lze provádět analýzu LoRaWAN rámců. Celé blokové schéma analyzátoru lze pak vidět na obrázku 4.2, kdy každý blok bude blíže popsán v následujících sekcích.



Obrázek 4.2: Blokové schéma analyzátoru

Prvním blokem analyzátoru je přijímač nacházející se v souboru **receiver.py**. Přijímač má na starosti vytvořit python objekt softwarově definovaného rádia, příjem rádiových vln a rozeznání užitečného signálu od šumu.

Je-li přijat nějaký užitečný signál je v dalším bloku předzpracován pro budoucí analýzu. Předzpracování zaručuje soubor **preprocessor.py**.

Po předzpracování užitečného signálu následuje demodulátor, kterého obsah je v souboru **demodulator.py**. Hlavní úlohou demodulátoru je určit činitel rozprostření přijatého signálu a získat z jednotlivých datových symbolů hodnoty, které reprezentují. Demodulátor zároveň i rozhoduje, zda je přijatý signál LoRaWAN rámcem. Dá se totiž předpokládat, že pokud nevyjde jeden z kroků v demodulátoru, nejedná se o LoRaWAN rámec.

Pokud prošly všechny kroky v demodulátoru, je zřejmé, že se jedná o LoRaWAN rámec, a tudíž je poslán dál v přenosovém řetězci do dekódéru vyskytujícího se v souboru **decoder.py**. Zde je na LoRaWAN rámec pro zrekonstruování původní zprávy postupně aplikováno Grayovo kódování,

inverzní operace k prokládání (`deinterleaving`), Hammingovo dekódování a interzní operace k bělení (`dewhitening`).

Posledním blokem v přenosovém řetezci je soubor **WiresharkStreamer.py**, který slouží jako rozhraní mezi Python implementací analyzátoru a softwarem Wireshark pro zprostředkování výsledných dat.

Spouštěný soubor za účelem analýzy LoRaWAN rámců je soubor **main.py**. V něm jsou importovány všechny funkce z předešlých bloků, které na sebe navazují a tím tvoří jeden funkční celek.

Podpůrnou úlohu vykonává soubor **config.py**. Obsahuje přehledně na jednom místě veškerou potřebnou konfiguraci využívanou ostatními bloky.

4.4 Příjem signálu

Za předpokladu, že jsou nainstalovány balíčky pro Adalm-Pluto SDR, lze jej připojit k zařízení, na kterém bude spouštěna implementace analyzátoru a využívat jej pro příjem LoRaWAN rámců.

4.4.1 Vytvoření rádia a zahájení příjmu

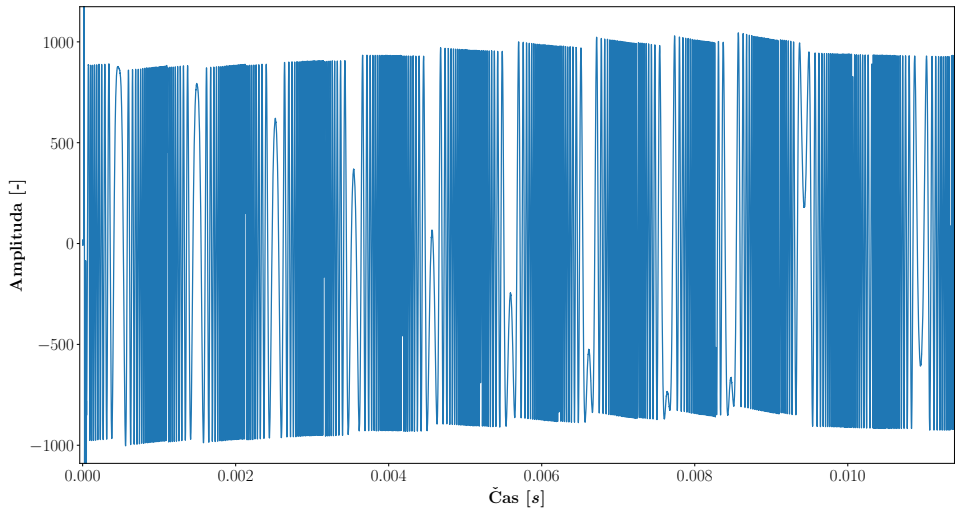
Pro vytvoření Python objektu rádia je nutné importovat pyadi-iio API do zdrojového kódu a poté jej lze vytvořit. V analyzátoru je objekt rádia vytvářen funkcí **createRadio()**. Naimportování i vytvoření Python objektu rádia lze vidět ve výpisu 4.1. Za účelem zachytávání LoRaWAN přenosů jsou výchozí hodnoty parametrů funkce `createRadio` definovány následně popsaným způsobem. Aby bylo možné zachytávat celé frekvenční pásmo dle zadání (867,0-870,0 MHz) je středová frekvence definována na frekvenci 868,5 MHz (`fc`), a tudíž velikost šířky pásma (`rxRfBw`) musí být 3 MHz. Je potřeba si uvědomit, že před navzorkováním, je signál pomocí směšovače převeden do základního pásma. Při použití šířky pásma 3 MHz je tedy nutné dodržet Nyquistův–Shannonův vzorkovací teorém, a proto je vzorkovací frekvence (`fs`) rovna 6 MHz. Velikost vyrovnávací paměti (`buffer`) je nastavena na přibližně 12,5 miliónů vzorků. Pro automatickou regulaci zisku byla jako výchozí hodnota zvolena textový řetězec *"fast_attack"*. Uvnitř funkce se vytvoří Python objekt rádia (`sdr`), kterému jsou předány parametry z funkce, a poté je objekt i s předanými parametry vrácen.

```
import adi

def createRadio(fs = 6e6, fc = 868.5e6, buffer = 1024*1024*12, rxRfBw = 3e6,
    control = "fast_attack"):

    # create radio
    sdr = adi.Pluto()

    # configure properties
```

```
    sdr.sample_rate = fs
    sdr.rx_lo = fc
    sdr.rx_buffer_size = buffer
    sdr.rx_rf_bandwidth = rxRfBw
    sdr.gain_control_mode_chan0 = control

    return sdr
```

Výpis 4.1: Import pyadi-iio API a vytvoření rádia

Po vytvoření objektu rádia lze zahájit příjem. Pro to se na vytvořený objekt zavolá funkce **rx()**. Adalm-Pluto SDR se naladí na zadanou frekvenci a začne na ní přijímat její okolí definované předanou šířkou pásma. Po naplnění vyrovnávací paměti je výstup zprostředkován Python části v podobě IQ vzorků, navzorkované o definované vzorkovací frekvence.

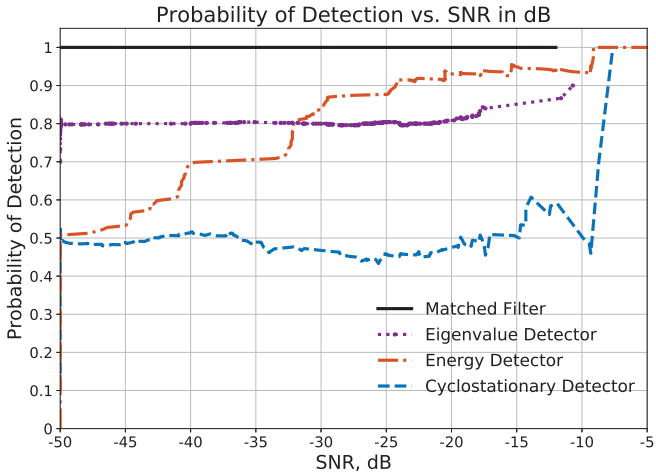


Obrázek 4.3: Příklad zachyceného signálu

4.4.2 Detekce užitečného signálu

Pro implementaci analyzátoru je rozeznání užitečného signálu od šumu jedna z nejdůležitějších oblastí. Není-li nic zachyceno, není ani možné nic analyzovat.

Způsobů pro detekci signálů je mnoho. Srovnání některých z nejpoužívanějších je obsaženo v zdroji dostupného na [21]. V něm jsou porovnány celkově 4 detekční metody - metoda přizpůsobeného filtru, metoda detekce výkonu, metoda *Eigenvalue Detector* (ED) a metoda *Cyclostationary*



Obrázek 4.4: Pravděpodobnost detekce signálu pro různé poměry signálu a šumu za využití různých detekčních technik [21]

Detector (CD). S odkazem na závěr zdroje a obrázek 4.4 jen metody přizpůsobeného filtru a detekce výkonu dosáhly 90% pravděpodobnosti detekce signálu při SNR -20 dB.

I přesto, že metoda přizpůsobeného filtru dosahuje lepších výsledků. V analyzátoru je použita metoda detekce výkonu - jedná se o spolehlivou metodu, která je jednoduchá na implementaci.

Pro její implementace je potřeba nejdříve stanovit meze, na základě kterých se bude rozlišovat užitečný signál od šumu. Výpočet mezí je znázorněno ve výpisu 4.2. Obsahem výpisu je funkce **getThresholds**, kde její parametr `received` jsou IQ vzorky z výstupu softwarově definovaného rádia (viz obrázek 4.3).

Pro zachycené vzorky se vypočítají absolutní hodnoty (`rAbs`) a z nich se zjistí nejvyšší hodnota (`rMax`) pro odvození meze pro určení užitečného signálu (`rHigh`). Poté je vypočítán průměr z absolutních hodnot (`rAvg`), který zároveň slouží jako mez pro stanovení šumu a zároveň je použit ve výpočtu pro mez `rHigh`. Ve výpočtu meze pro užitečný signál figuruje ještě konstanta `RECEIVE_THRESHOLD` nacházející se v souboru `config.py` (viz kapitola 4.3), s kterou lze mez zvyšovat, a nebo snižovat. Z funkce jsou vráceny jak obě meze, tak i absolutní hodnoty IQ vzorků pro následující detekci signálu.

```
def getThresholds(received):
    rAbs = abs(received) # get magnitudes
    rMax = max(rAbs) # get highest magnitude
    rAvg = sum(rAbs) / len(rAbs) # get average
    rHigh = ((rMax + rAvg) /
           RECEIVE_THRESHOLD) # set threshold for possible signal detection
    rLow = rAvg # set threshold for noise

    return rAbs, rHigh, rLow
```

Výpis 4.2: Stanovení mezí pro detekci užitečného signálu

Detekci užitečného signálu provádí funkce `detectSignal`. Jejím vstupem jsou IQ vzorky ze softwarově definovaného rádia (`received`) a minimální délka (`frameMinLen`), kdy IQ vzorky považovat za užitečný signál. Uvnitř funkce je volána funkce `getThresholds`, která vrací meze a absolutní hodnoty IQ vzorků. Poté jsou jednotlivé absolutní hodnoty procházeny for cyklem a pokud se narazí na vzorek, jehož velikost je větší než mez `rHigh`, označí se jako možný začátek užitečného signálu. Možný konec užitečného signálu značí vzorek, jehož velikost je menší než `rAvg`. Pokud rozsah vzorků od začátku do konce je větší než uživatelem definována minimální délka (`frameMinLen`), jsou vzorky považovány za užitečný signál a jsou předány do předzpracovacího bloku (viz kapitola 4.3).

4.5 Ověření proveditelnosti minimálního prototypu

Přijímač byl první implementovanou části analyzátoru. Kromě oprav případných vyskytnutých chyb v přijímači, a nebo změně parametrů pro vytvoření objektu rádia, zůstal zdrojový kód víceméně po celou dobu neměnný. Jelikož přijímač i inicializace rádia již byl popsán v předchozí kapitole, lze přejít k popisu, kdy bylo ověřeno, že lze implementovat LoRaWAN analyzátor, který bez žádných doplňujících informací je schopen provést sérii operaci ke korektnímu zachycení LoRaWAN rámce a jeho následné demodulaci a dekódování.

Pro testování funkčnosti a možné ladění byly z počátku z ACR-CV modulu (viz kapitola 4.10.1) zachyceny zprávy pro každý činitel rozprostření (7-12) s šířkou pásma 125 kHz a s užitečnými daty obsahující textový řetězec "testData." Jednotlivé zachycené zprávy se uložily do souborů s příponou `.npy`, které se pro případ otestování funkčnosti implementace načetly.

Prvotní velkou překážkou bylo, jak zjistit činitel rozprostření bez žádné dopňující informace. Z toho vyplývá, že se musela vytvořit nějaká operace, která by procházela přijatý signal pro všechny činitele rozprostření, a tak jej určila. Jelikož preambule obsahuje obvykle 8 up-chirp symbolů, dva modulované symboly a 2,25 down-chirp symbolů, Bylo prvotní ideou použít lokálně vygenerovaný up-chirp symbol pro každý činitel rozprostření a použít jej pro korelaci s preambulí případného LoRa signálu. Lze se tedy vydat dvěma cestami. Při první lze projít všechny činitele rozprostření a vybrat ten, který dosáhl v porovnání s ostatními nejvyšší míry korelace. Nicméně korelace je poměrně výpočetně náročná operace, kdy náročnost stoupá čím vyšší je činitel rozprostření přijatého signálu. A proto procházet všechny možné varianty by mohlo zabrat poměrně dost času. Druhou možností je procházet jeden činitel rozprostření po druhém, a pokud pro daný činitel rozprostření přesáhne míra korelace stanovenou mez, označit jej jako činitel rozprostření signálu. Zde ale neexistuje jednoznačný způsob, kdy lze říct, jaká mez je vhodná. Pokud by byla příliš nízká, je možné, že by byl stanoven činitel rozprostření, ale ve skutečnosti by byl vyšší - jen se k tomu kroku ještě nedošlo. Pokud

by byla mez vyšší, je možné, že pro znatelnou část přijatých signálů by činitel rozprostření nebyl stanoven.

Jelikož preambule obsahuje i 2,25 down-chirp symbolů, bylo další myšlenkou využít pro korelaci je. Za tímhle účelem byl lokálně vygenerován down-chirp symbol pro každý činitel rozprostření a provedena mezi ním a přijatým signálem korelace. Ovšem nastávají zde stejné problémy jako v předchozím případě. Navíc v předchozích možnostech bylo možné provést korelaci pro pár prvních up-chirp symbolů a přestat. Zde taková možnost neexistuje.

I když žádná z popsaných metod nebyla zcela vhodná, bylo tak možné ověřit, že lze zjistit činitel rozprostření, a tudíž pokračovat v demodulaci a dekódování přijatého signálu.

Standardním postupem pro demodulaci je vynásobit jednotlivé symboly s down-chirp symbolem o zjištěném činiteli rozprostření, kdy jednotlivé přijaté symboly se převedou na průběhy o konstantní frekvenci (viz kapitola 4.1), z kterých lze Fourierovou transformací získat hodnoty symbolů. Proto bylo provedeno pronásobení a ověřeno, že z průběhu by se daly získat jednotlivé hodnoty symbolů.

Jelikož byla ověřeno, že je možné určit činitel rozprostření bez žádných doplňujících informací, i že se dá provést demodulace, byl v dalším pořadí dekódér. Jenže dekódovací část je hojně popsána v již popsaných implementacích (viz kapitola 4.1), a tudíž její ověřování nebylo potřeba.

V návaznosti na zmínku o kapitole 4.1 lze podotknout, že ve zmíněných zdrojích je převážně používán známý činitel rozprostření. Díky tomu je možné vygenerovat down-chirp symbol, vynásobit s celým přijatým signálem, detekovat preambuli na základě Fourierovy transformace a provést demodulaci s dekódováním. I tahle metoda byla vyzkoušena, ale pronásobení by se muselo udělat pro každý činitel rozprostření. Nelze totiž dostat očekávané průběhy, pokud jsou rozdílní činitelé rozprostření pro down-chirp symbol a přijatý signál. V případě použití této metody, kdy by se iterovalo mezi všemi eventualitami, je možné, že výpočetní náročnost by mohla být vyšší než v případech s korelací.

4.6 Předzpracování přijatého signálu

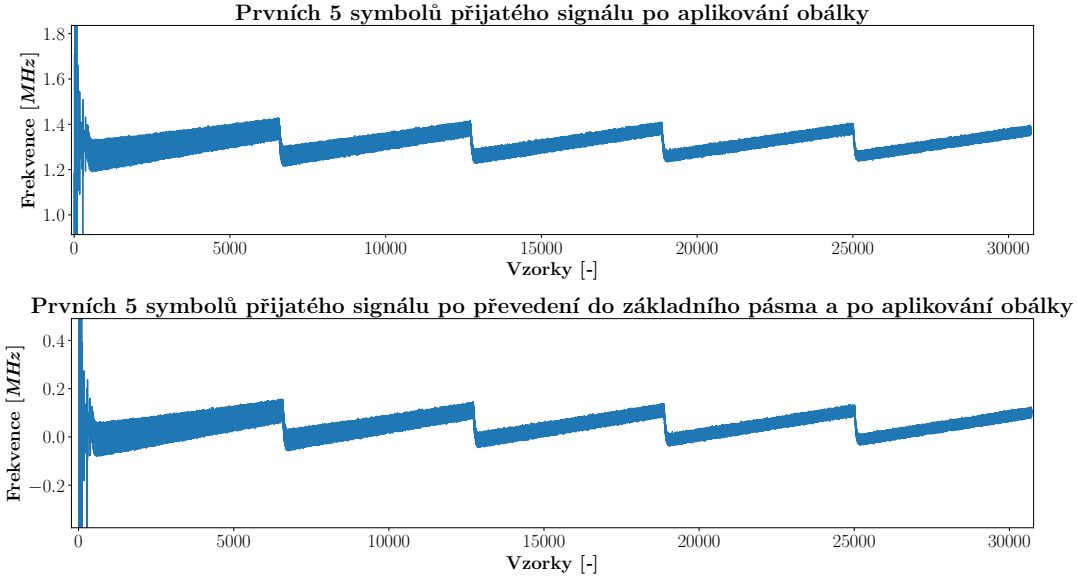
Úlohou předzpracování je přichystat přijatý signál tak, aby byl vhodnější pro další zpracování. Jelikož lze naslouchat na poměrně širokém pásmu (3 MHz), můžou se zde objevovat signály o vyšších frekvencích než je softwarově definované rádio naladěno, a proto je přijatý signál převeden do základního pásma. Kvůli velké šířce pásma je i vysoká vzorkovací frekvence (6 MHz). Je poměrně výpočetně náročné zpracovávat signály o tak vysoké vzorkovací frekvence, a proto jsou pro snížení výpočetní náročnosti s ohledem na zjištěnou šířku pásma podvzorkovány.

4.6.1 Převedení přijatého signálu do základního pásma

I když softwarově definované rádio dělá své posunutí do základního pásma, jsou vlivem velké šířky pásma některé signály přijaty s frekvenčním odstupem od naladěné frekvence. Nejde o chybu, nýbrž

je třeba si uvědomit, že pokud je zařízení naladěno na definovanou frekvenci a příjme signál pod, nebo nad definovanou frekvenci v definované šířce pásma, má přijatý signál jinou frekvenci z pohledu naladěné frekvence.

Situaci lze lépe vidět na obrázku 4.5, kdy softwarově definované rádio bylo naladěno na frekvenci 868,5 MHz. Využitím ACR-CV modulu byla zaslána zpráva na frekvenci 869,5 MHz a přijatá analyzátorem. Na přijatém průběhu (na obrázku nahoře) se dá vidět, že kmitá kolem frekvence 1 MHz. Nicméně po aplikování níže uvedeného postupu lze pozorovat změnu ve frekvenční oblasti (na obrázku dole), kdy se přijatý průběh již nachází kolem 0 MHz (=převeden do základního pásma).



Obrázek 4.5: Prvních 5 symbolů přijatého signálu před převedením (nahoře) a po převedení do základního pásma (dole)

Za účelem převedení přijatého signálu do základního pásma je nejdříve zjištěna jeho nosná frekvence. Její určení se provádí ve funkci `detectCarrierFrequency` (viz výpis 4.3). Parametry funkce jsou přijatý signál (`sig`), vzorkovací frekvence (`fs`), odstup od začátku/konce signálu (`offset`) a rozsah vzorků, z kterých se má nosná frekvence určit (`size`).

Pro parametr `offset` je definována konstanta `T_OFFSET` v souboru `config.py`, která říká, že se má přeskočit 500 prvních/posledních vzorků. Odstup od začátku/konce signálu je zde definován, protože se při analýze přijatých signálů ukázalo že na jejich začatcích i koncích bývá nevyžádaná část (přechodový jev), která s užitečným signálem nijak nesouvísí (patrná například na začátku signálu na obrázku 4.5). Původ téhle části zatím stále není známý - je možné, že ji nějakým způsobem generuje SDR při příjmu, popřípadě původ je už na vysílací straně. Tahle hodnota byla určena na základě pozorování zachycených signálů, a tudíž by se mělo jednat o bezpečnou hranici pro velkou

většinu signálu. Nejedná se sice o zaručené řešení, a proto by se v budoucnu mohla vymyslet operace, která by danou nevyžádánou část odstraňovala, popřípadě přijít na příčinu a tu odstranit.

Parametr `size` je další konstantou, kterou lze v `config.py` souboru nalézt pod konstantou `CARRIER_SIZE`. Rozsah je zde definován, protože nosná frekvence se dá poměrně spolehlivě určit i z malého rozsahu vzorků, čímž se ušetří výpočetní náročnost. Nicméně volba rozsahu je dle uvážení a v případě potřeby je možné určit nosnou frekvenci i z celého přijatého signálu nastavením parametru na hodnotu menší než nula.

Uvnitř funkce je nejdříve zkontrolováno, jestli předaný rozsah nepřesahuje délku pole IQ vzorků a zda je předán validní rozsah vzorků pro stanovení nosné frekvence. Pokud se ukáže jedna z podmínek jako validní, je rozsah upraven, tak aby pokrýval celý přijatý signál kromě nevyžádané části na konci přijatého signálu.

Poté je využitím knihovny `numpy` provedena na definovaný rozsah vzorků rychlá Fourierova tranformace. Počet frekvenčních složek ve spektru je dán parametrem `size`, kde krok (v hertzích) mezi nimi je přibližně $f_s/size$. Na základě toho lze ze spektra vyvodit jednotlivé frekvence vztahem:

$$f_n = \frac{n \cdot f_s}{size} \tag{4.1}$$

Kde f_n je hledaná frekvence, n je k. frekvenční složka ve spektru, f_s je vzorkovací frekvence a $size$ velikost FFT.

Z výstupu rychlé Fourierovy transformace je vzat index nejsilnější složky ve spektru (=nosná frekvence) a vypočítána nosná frekvecne přijatého signálu dle vztahu 4.1.

```
def detectCarrierFrequency(sig, fs, offset, size):
    e =  offset + size
    if (e > (len(sig) - offset)) or (size <= 0): # if invalid size - correct it
        e = len(sig) - offset
        size = e - offset

    # FFT calculation
    Sxx = np.fft.fft(sig[offset : e])

    # Find index with highest value
    idx = np.argmax(np.abs(Sxx))

    # Calculate carrier frequency and return it
    return (idx * fs) / size
```

Výpis 4.3: Určení nosné frekvence přijatého signálu

Následně je celý přijatý signál vynásoben s vygenerovaným sinusovým signálem o vypočítané nosné frekvenci, čímž se převede do základního pásma.

4.6.2 Určení šířky pásma

Celkově se pro LoRaWAN signály používají tři různé šířky pásma - 125 kHz, 250 kHz a 500 kHz. Proto se po každou šířku pásma a vzorkovací frekvenci SDR vypočítá velikost (ve vzorcích) jednoho SF=12 symbolu (`samps`). Ze stejného důvodu jako při určení nosné frekvence se definuje odstup od začátku přijatého signálu (`offset`), ke kterému se přičte polovina velikosti SF12 symbolu - tím vznikne nově definovaný odstup (`new_offset`). Šířka pásma se následně určuje ze vzorků od `new_offset` do `new_offset + samps`, čímž by se mělo zaručit, že pro každý činitel rozprostření je v daném rozsahu obsažen aspoň jeden peak.

Následně je na vybrané vzorky aplikována obálka (viz například obrázek 4.6), z které se zjistí nejvyšší a nejnižší kmitočet. Pokud se "vzdálenost" v hertzích mezi oběma kmitočty nachází poblíž jedné ze tří šířek pásem, je předpokládáno, že se jedná o skutečnou šířku pásma zachyceného signálu.

Na základě šířky pásma je s ohledem na dodržení Nyquistova–Shannonova vzorkovacího teorému zvolena pro následující krok podvzorkovací frekvence specifikována v konstantě `DOWNSAMPLING_FREQUENCIES` v souboru `config.py`.

4.6.3 Podvzorkování přijatého signálu

Podvzorkování, nebo taky snížení počtu vzorků přijatého signálu se řeší ve funkci `downSampleSignal` (viz výpis 4.4). Vstupem je přijatý signál převeden do základního pásma (`down`), vzorkovací frekvence použitá při vytváření rádia (`fs`) a podvzorkovací frekvence (`fsdown`). Uvnitř funkce je vypočítáno, jak moc se daný přijatý signál má podvzorkovat, a poté se provede podvzorkování funkcí `decimate` z knihovny `scipy`. V ní byl zvolen filtr typu FIR, který oproti výchozímu IIR filtru vykazoval lepší chování při pozdějším zpracování.

```
def downSampleSignal(down, fs, fsdown):
    # Calculate how much will be signal downsampled
    ratio = int(fs // fsdown)

    # Downsample signal and return it
    return signal.decimate(down, ratio, ftype="fir")
```

Výpis 4.4: Podvzorkování přijatého signálu

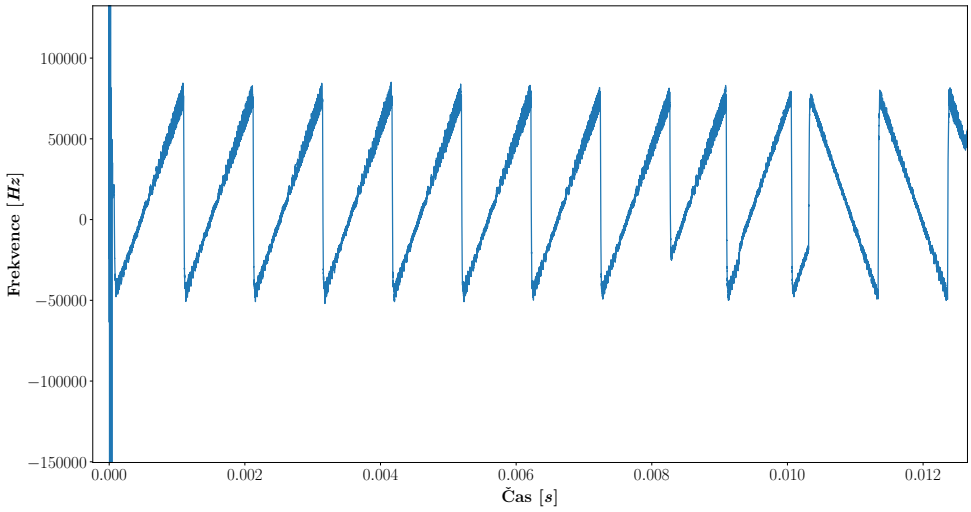
Dlouho také byla používána funkce `resample` z knihovny `scipy`, která vykazovala podobné chování jako funkce `decimate`. Nicméně fukce `resample` využívá při podvzorkování Fourierovu transformaci, kdy při delších signálech byla výpočetní náročnost znatelná. V některých případech bylo zpozorováno, že funkce `decimate` byla až 10x rychlejší než funkce `resample`.

4.7 Demodulace přijatého signálu

Jak již bylo pospáno v kapitole 4.5, většina řešení pro stanovení činitele rozprostření nebyla vhodná. Za tímhle účelem byla vyvinuta metoda, která hledá periodu up-chirp symbolů v preambuli a na základě ní určí činitel rozprostření. Metoda je blíže popsána v podkapitole 4.7.1. Zíkání hodnot jednotlivých symbolů je blíže popsána v podkapitole 4.7.3, kde je popsána momentálně používaná metoda i její alternativa, která by se mohla v budoucnu použít.

4.7.1 Stanovení činitele rozprostření

Preambule LoRa signálu se obvykle skládá z osmi up-chirp symbolů, dvou modulovaných symbolů a dva a čtvrt down-chirp symbolů (viz obrázek 4.6). Pro určení činitele rozprostření (Spreading Factor) jsou využity up-chirp symboly, respektive vzdálenosti mezi jednotlivými peaky/konci symbolů. Jelikož pro různé činitele rozprostření jsou různé délky symbolů, lze na základě periodicity peaků určit SF signálu.



Obrázek 4.6: Preambule přijatého LoRa signálu po aplikování obálky

Metoda určení periody je založená na myšlence, že z podstaty up-chirp symbolu musí být mezi posledními vzorky symbolu a prvními vzorky následujícího symbolu velký frekvenční rozdíl. Pokud se tento rozdíl nalezne, lze předpokládat, že byl nalezen peak.

Pro detekci peaků slouží funkce `findSignalSF`, kde jejími parametry jsou: přijatý signál (`y`), momentálně používána vzorkovací frekvence (`fSample`) a šířka pásma (`bw`). Za účelem detekce konců symbolů je definována mez, která rozhoduje, zda frekvenční rozdíl je dostatečný, respektive jestli je frekvenční rozdíl pod stanovenou mezí (viz obrázek 4.8). Pro vypočítaní meze je vybrán rozsah

vzorků (s odstupem od začátku signálu), který odpovídá uživatelem volitelné velikosti SF7 symbolu (velikost by měla být volena tak, aby neobsahovala případný peak pro pozdější vypočítané průměrné hodnoty).

Vybraný rozsah vzorků se předá funkci `getSFDetectionThreshold`, kterou lze vidět ve výpisu 4.5. Parametry jsou: vybraný rozsah vzorků (`y`), koeficient (`coeff`), s kterým lze mez zvyšovat, a nebo snižovat. S posledním parametrem funkce (`range`) je možné určit, kolik nejmenších hodnot se vezme pro výsledný výpočet meze.

```
def getSFDetectionThreshold(y, coeff, range = 0.1):
    diff = np.diff(freqInTime(y), 1)
    avgSig = np.sum(diff) / len(diff) # calculate average from given bunch of
        samples
    env = np.abs(diff) # get magnitudes

    envMins = sorted(env)[-(int(len(env) * range)):] # select minimums given to
        the range

    avgMin = sum(envMins) / len(envMins) # get average from selected minimums

    threshold = avgSig - (avgMin * coeff) # actual threshold

    return threshold
```

Výpis 4.5: Vypočítání meze pro detekci peaků

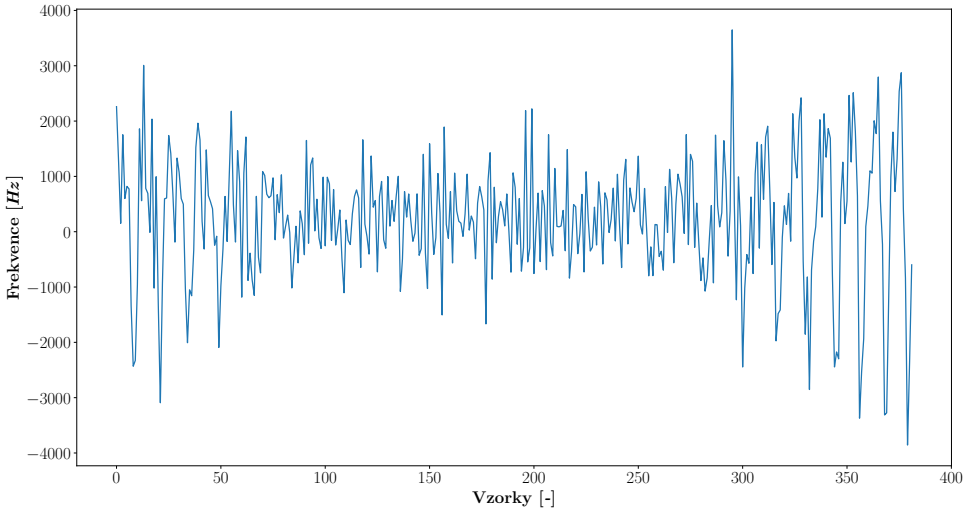
Na vybrané vzorky je ve funkci aplikována obálka (funkce `freqInTime`) a na ní jsou vypočítány rozdíly mezi jednotlivými vzorky, kdy rozdíl mezi dvěma vzorky je vypočítán jako:

$$out[i] = a[i+1] - a[i] \tag{4.2}$$

Znamená to tedy, že se průběh převede do podoby, kdy frekvence je závislá na čase a na něm jsou hledány frekvenční rozdíly mezi dvěma vzorky. Po aplikování operace se vytvoří například průběh, jaký je možné sledovat na obrázku 4.7.

Z průběhu se vypočítají absolutní hodnoty, z kterých se na základě předaného parametru `range` najde definovaný počet nejvyšších hodnot (`envMins`) značících nejvyšší variace mezi jednotlivými vzorky. Ze získaných hodnot (`avgMin`) i z celého průběhu (`avgSig`) se vypočítají průměry, které jsou použity spolu s koeficientem pro výpočet meze pro detekci peaků (`threshold`).

Je nutné si uvědomit, že v tomhle případě nelze použít jako mez pouze průměr z celého průběhu (`avgSig`), jelikož by se velké množství vzorků mohlo nacházet pod stanovenou mezí, a tudíž by mohlo dojít k ztížení detekce jednotlivých peaků. Proto je pro stanovení meze přidán i průměr z



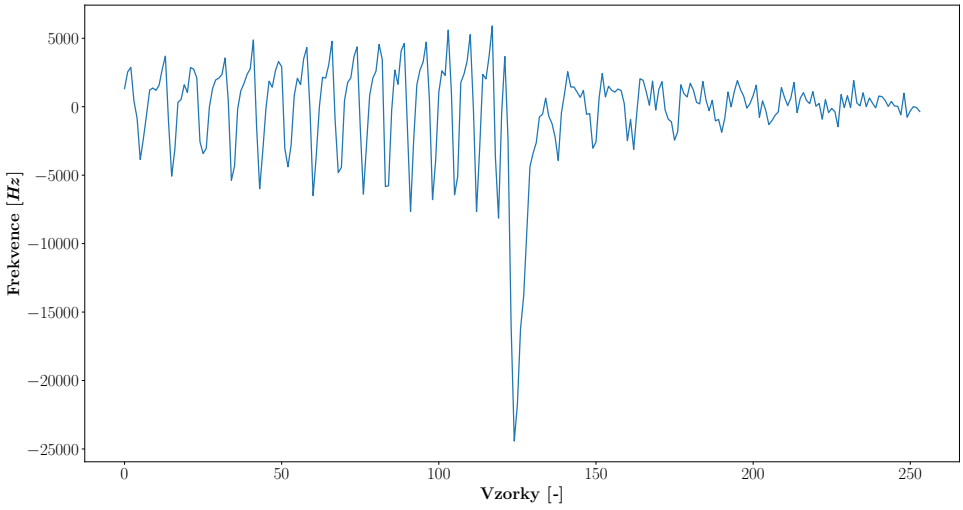
Obrázek 4.7: Rozdíly mezi vzorky pro stanovení meze

největších variací (`avgMin`) společně s koeficientem (`coeff`), na základě kterých by mělo být většina vzorků vyfiltrováno a tím zpřesnit detekci peaků.

Po stanovení meze se pro každý činitel rozprostření skáče do míst, kde by se měl nacházet konec symbolu. V místech skoků je vybrána oblast vzorků, na které se (stejně jako při stanovení meze) aplikuje obálka a vypočítají rozdíly mezi nimi. Ve vzniklém průběhu (viz obrázek 4.8) se prochází jeden vzorek po druhém a hledají se oblasti pod vypočítanou mez. Je-li nějaká taková lokální oblast nalezena, je vybrán vzorek s nejnižší hodnotou. Ze všech získaných nejnižších hodnot z lokálních oblastí se následně určí nejnižší, která by měla znázorňovat konec symbolu (na obrázku 4.8 se jedná o cca 124. vzorek). Protože je v preambuli obvykle osm up-chirp symbolů, je peak hledán celkově 8x.

Při každém dalším skoku se aktualizuje pozice a rozsah oblasti, ve které další konec symbolu hledat. První skok totiž může být celkem nepřesný, a proto je i prohledávána oblast dost velká. Ta se aktualizuje na základě toho, jak moc se detekce peaků zpřesňuje. Je-li detekce peaků přesnější, prohledávaná oblast se zmenšuje. V opačném případě se oblast zvětšuje. Nicméně v obou případech nemůže oblast přesáhnout určité velikosti. Nejmenší velikost funguje jako pojistka pro vždy bezpečný počet vzorků, v kterých peak hledat, v opačném případě, aby prohledávána oblast nepřesáhla do dalšího symbolu. V takovém případě by mohly být v prohledávané oblasti dva konce symbolů, kdy by se detekoval pouze jeden z nich, čímž by docházelo k chybě při určení činitele rozprostření.

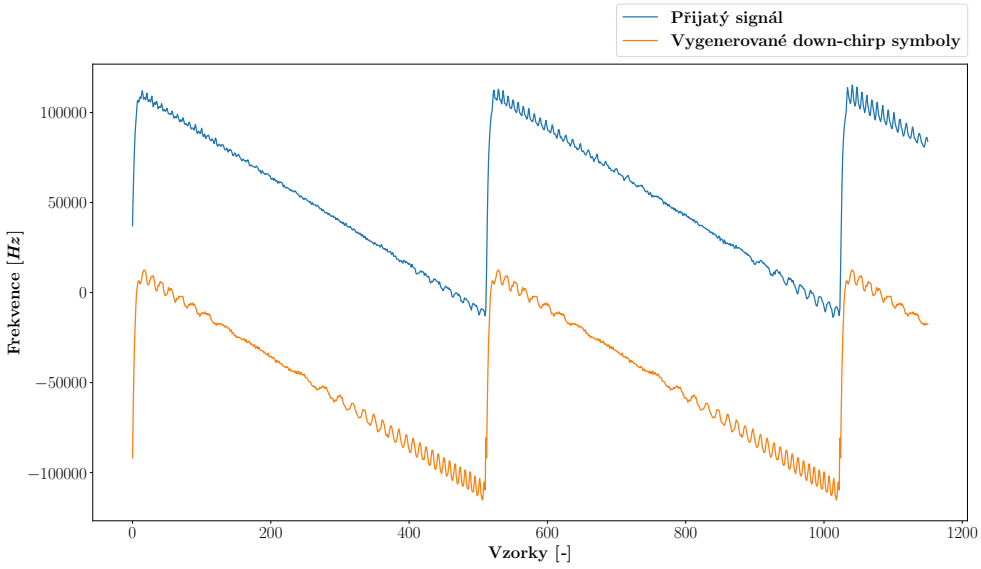
Během hledání peaků je také počítána perioda. Vypočítá se jako vzdálenost ve vzorcích mezi posledním koncem symbolu a nyní nalezeným. Pokud se v předchozím kroku peak nedetekoval, počítá se vzdálenost ve vzorcích mezi nyní nalezeným peakem a předpokládanou polohou předchozího konce symbolu.



Obrázek 4.8: Prohledáváná oblast pro detekci peaku

Jakmile se projde všech 8 up-chirp symbolů v preambuli pro všechny činitele rozprostření, jsou vybráni kandidáti pro činitel rozprostření signálu na základě nejvyššího počtu detekovaných peaků, kdy kandidátů může být víc. I když by měla mez (`threshold`) vyfiltrovat většinu falešných detekcí, může se stát, že spolu se skutečným činitelem rozprosření signálu má stejný počet detekovaných peaků i jiný činitel rozprostření. Proto na základě odvozených vzdáleností (period) od peaků je vypočítána pro každého kandidáta pro SF signálu konečná perioda, která musí být v definovaných mezích. Pokud není, nepředpokládá se, že se jedná o činitel rozprostření pro daný signál. Tím se tedy může vyfiltrovat pár falešných detekcí. Nicméně rozhodující podmínkou pro určení činitele rozprostření jsou až down-chirp symboly.

Pro všechny činitele rozprostření s validní periodou je vzat poslední známý up-chirp symbol z preambule. Z něj se vytvoří komplexně sdružený symbol (= downchirp). Z jednoho down-chirp symbolu se vytvoří 2,25 down-chirp symbolů a od posledního známého indexu peaku se provádí korelace mezi vygenerovanými down-chirp symboly po aplikování obálky a přijatým signálem po aplikování obálky. Obálky jsou zde použité, protože míra korelace byla vyšší v porovnání s případem, kdy použita nebyla. Korelace se provádí s krokem zjištěné periody až do pozice, kde by se v preambuli měly nacházet down-chirp symboly. Skutečný činitel rozprostření daného přijatého signálu je určen na základě nejvyšší zjištěné míry korelace. Příklad, jak vypadají oba průběhy, mezi kterými se provádí korelace lze vidět na obrázku 4.9.



Obrázek 4.9: Korelace mezi přijatým signálem a vygenerovanými down-chirp symboly

4.7.2 Stanovení činitele rozprostření pro downlink

Výše popsaná metoda se vztahuje k uplink zprávám. Nicméně u koncentrátorů je zvykem používat IQ invertovaný signál. To znamená, že prvních 8 symbolů jsou down-chirp symboly, pak dva modulované down-chirp symboly, dva a čtvrt up-chirp symbolů a nakonec down-chirp modulované datové symboly.

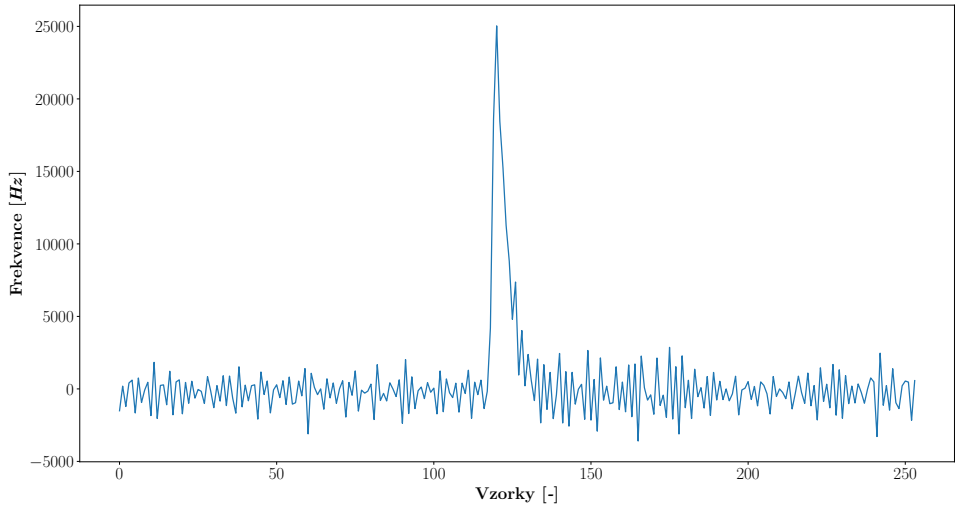
Pokud se nedokáže určit činitel rozprostření napoprvé, je vyzkoušena varianta, zda se nejedná o downlink zprávu. Celý postup je úplně stejný kromě jedné výjimky. Když se aplikuje obálka na oblast pro detekci peaku, vypadá jako na obrázku 4.10. Z toho důvodu je z takového průběhu vytvořen průběh komplexně sdružený, čímž vznikne průběh jako na obrázku 4.8, a proto lze ponechat všechny ostatní postupy.

Nevadí to ani při korelaci, protože se vždy bere komplexně sdružený symbol k poslednímu známému symbolu při detekci peaků. Jelikož je tento komplexně sdružený vrácen, nevadí to ani při získání hodnot jednotlivých symbolů (viz kapitola 4.7.3).

Nicméně ve zbytku práce se bude pro jednoduchost počítat s variantou, že je použita uplink zpráva.

4.7.3 Získání hodnot jednotlivých symbolů

Pro získání hodnot symbolů se jednotlivé symboly vynásobí s již vygenerovaným down-chirp symbolem. V takovém případě by se měl průběh, jenž symbol reprezentuje, převést na průběh o konstantní frekvenci, z kterého lze pomocí rychlé Fourierovy transformace a přepočtu získat skutečnou hodnotu symbolu.



Obrázek 4.10: Prohledáváná oblast pro detekci peaku pro downlink

Proto je také jako down-chirp symbol používán komplexně sdružený průběh poslednímu up-chirp symbolu z preambule. I když down-chirp symboly generované pomocí knihovny `scipy`, a nebo lokálně dle matematického vyjádření chirp signálů měly dokonalý tvar, více korektně demodulovaných hodnot dosahoval komplexně sdružený průběh.

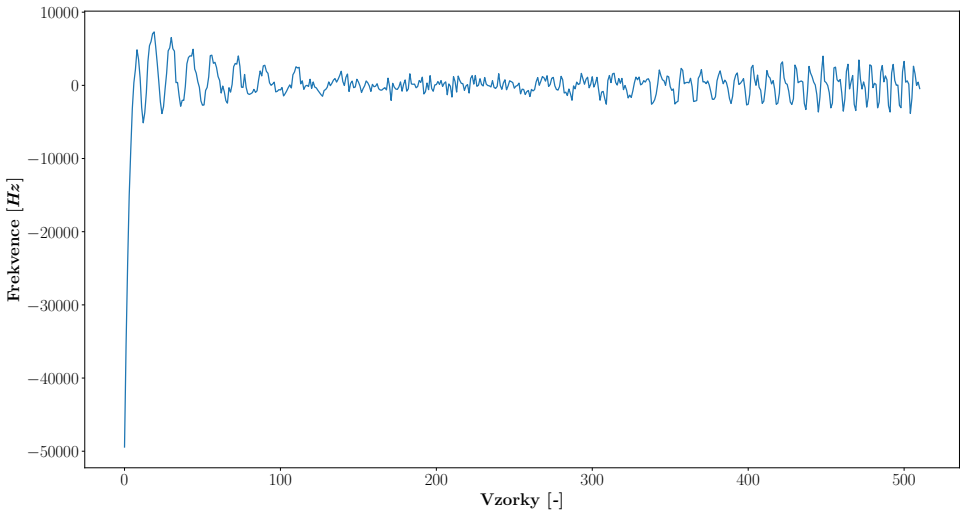
Pro získání hodnot jednotlivých symbolů je nejdříve vynásobeno prvních osm up-chirp symbolů v preambuli s vygenerovaným down-chirp symbolem ve funkci `dechirpSignal`. Ve funkci jsou procházeny jednotlivé symboly, kdy každý je vynásoben s down-chirp symbolem a poté jsou vráceny vynásobené průběhy vráceny (lze vidět na obrázku 4.11).

Kvůli možné mírné desynchronizace, nebo přechodovým jevům down-chirp symbolu je z vytvořeného průběhu vzato pouze definovaný rozsah vzorků. Příkladem přechodového jevu down-chirp symbolu může být prvních pár vzorků na obrázku 4.9. Důsledkem toho vznikne na začátku pronásobeném průběhu přechodový jev (viz obrázek 4.11), který by mohl mít vliv na stanovení hodnoty symbolu. Jak velký rozsah se vezme je definováno konstantou `FFT_DISTANCE`, kdy se odebere definované procento vzorků ze začátku i konce pronásobeného průběhu. Příklad použití konstany `FFT_DISTANCE` lze vidět například ve výpisu 4.6.

Na vybraný rozsah vzorků se použije rychlá Fourierova transformace, kde její velikost je zvětšena kvůli větší přesnosti. Kolikrát je velikost větší je definováno konstantou `FFT_FACTOR`. Proces je opakován pro každý up-chirp symbol, kdy se pokaždé z výstupu transofrmace vezme pozice nejsilnější frekvenční složky. Na konci procesu je vrácen index s nejvyšším počtem výskytu.

Po preambuli jsou i jednotlivé datové symboly vynásobeny s vygenerovaným down-chirp symbolem. V tomhle případě vznikne průběh, který může vypadat jako na obrázku 4.12.

Podoba průběhu je závislá na modulované hodnotě, tudíž pro různé hodnoty vypadá průběh

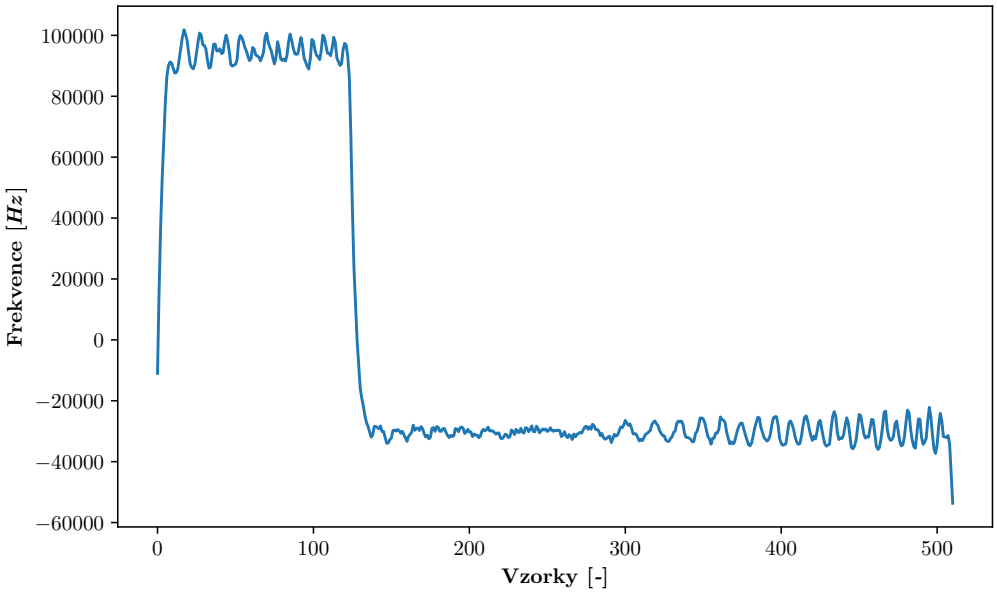


Obrázek 4.11: Up-chirp symbol po vynásobení s down-chirp symbolem

různě. V ideálním případě by měl vzniknout průběh o jedné dominantní frekvenci (viz například [17], [16]). Nicméně v případě téhle implemetace se ukázalo, že vznikne signál o dvou dominantních frekvencích (viz obrázek 4.12). Dlouho to představovalo překážku, kdy to vypadalo, že je něco prováděno nekorektně. Po dlouhém zkoumání ale bylo zpozorováno, že obě frekvence představují totéž. Jen je potřeba frekvence kruhově posunout, respektive použít zbytek po dělení (modulo) se šířkou pásma LoRa signálu.

Získání hodnot z datových symbolů režíruje funkce `getSymbolValues` (viz výpis 4.6). Vstupem funkce jsou datové symboly (`y`), perioda signálu (`period`), zjištěný index preambule z předchozího kroku (`premIdx`), kolikrát se má zvětšit velikost rychlé Fourierovy transformace (`fftFactor`) a činitel rozprostření signálu (`sf`).

```
def getSymbolValues(y, period, premIdx, fftFactor, sf):
    start = 0
    end = start + period
    values = []
    N = fftFactor*period
    premIdx = premIdx / fftFactor

    while end <= len(y):
        sym = y[start + int(FFT_DISTANCE * period) : end - int(FFT_DISTANCE *
            period)]
        Sxx = np.fft.fft(sym, n=N)
```



Obrázek 4.12: Datový symbol po vynásobení s down-chirp symbolem

```
    idx = np.argmax(np.abs(Sxx))
    value = int(((idx/fftFactor) - premIdx) % 2**sf) # calculate value
    values.append(value)
    start += period # move by symbol
    end += period

return values
```

Výpis 4.6: Získání hodnot z datových symbolů

Uvnitř funkce jsou procházey všechny datové symboly, respektive definovaný rozsah vzorků (ze stejného důvodu jako v případě preambule), z kterých je vybrána pozice nejsilnější frekvenční složky z výstupu rychlé Fourierovy transformace (`idx`). Výsledná hodnota symbolu je výpočítána dle vztahu (`value`):

$$value = int(((idx/fftFactor) - premIdx) \tag{4.3}$$

Jelikož pro přesnost byla zvětšena velikost FFT, je potřeba výslednou hodnotu zkorigovat, a proto je dělena hodnotou `fftFactor`. Zjištěná pozice preambule (`prem_idx`) se používá jako reference hodnoty 0. V ideálním případě by se měl up-chirp symbol (hodnota 0) po pronásobení s down-chirp symbolem nacházet na 0 Hz. Nicméně vlivem násobení s down-chirp symbolem, nebo

například při převedení do základního pásma mohlo dojít k mírnému posunu na jinou frekvenci. Tudíž pro korektní stanovení hodnot symbolu je nutné využít tuto referenci. Jelikož datový symbol může obsahovat hodnoty 0 až $2^{SF} - 1$, je na něj aplikován i zbytek po dělení příslušného činitele rozprostření.

Nutno podotknout, že myšlenka vynásobení signálu s down-chirp symbolem, využít FFT i výsledný přepočet byl inspirován převážně z [16]. Nicméně zmíněná myšlenka je standartem pro demodulaci LoRa signálu, kdy je využívána i v dalších implementacích jako [17],[7],[15],[18].

V rámci implementace byla vyzkoušena i vlastní myšlenka pro stanovení hodnot datových symbolů. Pro vytvoření průběhu o jedné dominantní frekvenci byla na pronásobené datové symboly aplikována obálka a zbytek po dělení se šířkou pásma. Z obálky se zjístila průměrná frekvence průběhu, z které byla stanovena hodnota symbolu.

Metoda ale byla náchylná na symboly, které měly průběh okolo 0 Hz. Jelikož frekvence pronásobených symbolů je celkem proměnlivá, stávalo se, že vzorky symbolů okolo 0 Hz přesahovaly tuto hranici a po následném kruhovém posunutí zanášely chybu. Jinak řečeno existovaly vzorky, které měly být nad 0 Hz, ale nacházely se pod hranicí, byly kruhově posunuty, a tudíž jejich frekvence byla okolo šířky pásma a naopak. Vlivem zanesených chyb docházelo k nekorektnímu určení skutečných hodnot datových symbolů.

Na druhou stranu, myslím si, že uvedený přístup by mohl být v konečném důsledku přesnější než použít FFT. Proto bude zmíněna metoda předmětem budoucího zkoumání.

4.8 Dekódování přijatého signálu

Zpočátku byl řešen dekódér na základě zdroje dostupného na [16]. Nicméně řešení se nepovedlo zprovoznit, a proto se přešlo k zhotovení dekódéru na základě github projektu k zdroji [7], respektive čerpáno z [22].

V rámci dekódovací části demodulované hodnoty postupně projdou Grayovým kódováním, inverzní operaci k prokládání (`deinterleaving`), Hammingovým dekódováním a inverzní operaci k bělení (`dewhitening`).

4.8.1 Grayovo kódování

V grayově kódu jsou sousedící hodnoty kódů (binárních čísel) navzájem odděleny pouze jedním bitem, což zabraňuje nechtěným chybám v přenosu dat. Díky této vlastnosti je hojně využíván v bezdrátových komunikacích, kde pravděpodobněji dojde k nesprávné interpretaci symbolu se symbolem sousedním než se symbolem náhodným. [22] V pythonu jej lze implementovat jako ve výpisu 4.7, kdy proces spočívá v aplikování XOR operace mezi předanou decimální hodnotu a jejím výsledkem po posunutí o jeden bit doprava.

```
def gray(val):
    return val ^ (val >> 1)
```

Výpis 4.7: Grayovo kódování

4.8.2 Deinterleaving

Pro LoRa zprávy se používá diagonální prokládání, kdy lze kvůli nekorektnímu určení hodnoty symbolu rozložit chybu do více kódových slov. Jednotlivé chybové bity je poté možné při Hammingovém dekódování opravit. [22]

K zrekonstruování proložených bitů slouží funkce `deinterleave` (viz výpis 4.8), kde jejím vstupem jsou hodnoty z výstupu grayova kódování (`grayed`), činitel rozprostření (`sf`) a délka kódového slova (`cwLen`). Uvnitř funkce se inicializuje pole o velikosti předaného činitele rozprostření a na každou vytvořenou pozici se vytvoří pole o velikosti kódového slova (`deinterleaved`), kde budou uchovány výsledné kódová slova v podobě binárních hodnot. Na to jsou jednotlivé decimální hodnoty z grayova kódování převedeny do polí, reprezentující jejich binární podobu (`grayedBin`). Následně jsou dle vztahu 4.4 vytvořena z grayových hodnot výsledná kódová slova, která jsou uložena do pole `deinterleaved`.

$$D_{(i-j-1)\%SF),i} = G_{i,j} \; ; \tag{4.4}$$

kde G je pole `grayedBin` a D je pole `deinterleaved`.

```
def deinterleave(grayed, sf, cwLen):
    # Pre-create array to store codewords
    deinterleaved = []
    for f in range(sf):
        deinterleaved.append([0] * cwLen)

    # Convert integers to binary list
    grayedBin = []
    for g in grayed:
        grayedBin.append(intToBinaryList(g, sf))

    # Do the deinterleaving
    for i in range(cwLen):
        for j in range(sf):
            pos = (i - j - 1) % sf
            deinterleaved[pos][i] = grayedBin[i][j]
```

```
    return deinterleaved
```

Výpis 4.8: Deinterleaving

Jinak řečeno - z hodnot z grayova kódování se vytvoří matice (`G`) o velikosti **CR x SF** (CR řádků a SF sloupců), kdy bity kódových slov v matici `G` jsou proložené následně popsanou logikou. Je předpokládáno, že pozice v matici jsou značeny `[i, j]`, kde `i` značí řádek a `j` značí sloupec. Na hlavní diagonále je vždy poslední kódové slovo (jeho počáteční bit je tedy na pozici [0, 0]). Další kódová slova pak začínají na následujícím sloupci nultého řádku. To znamená, že předposlední slovo začíná na pozici [0, 1], předpředposlední na pozici [0, 2], a tak dále, kdy počáteční bit prvního kódového slova je na pozici [0, SF - 1]. Pro každou další pozici následujícího bitu kódového slova se o jedno inkrementuje řádek i sloupec předchozího bitu ([i + 1, j + 1]). Stane-li se, že dojde k "přetečení" matice, respektive další pozice je mimo matici `G`, pokračuje se na následujícím řádku na nultém sloupci. Takhle se chodí po diagonálach, dokud počet bitů kódového slova odpovídá použitému kódovacímu poměru (je-li použit například kódovací poměr CR=4/5, kódové slovo bude mít velikost pěti bitů, pro CR=4/6 6 bitů, a podobně). Výsledná matice `D` by měla mít rozměry **SF x CR** (SF řádků a CR sloupců), a tedy výsledkem je SF kódových slov o velikosti CR.

Příkladem můžou být hodnoty [20, 11, 16, 8, 0, 22, 0, 5], kdy SF=5 a CR=4/8. Matice G bude uspořádána následovně:

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{4.5}$$

Na stejné matici se vynesou diagonály. Jelikož je použit CR=4/8, kódová slova budou 8 bitová.

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{4.6}$$

Z vynesených diagonál lze odvodit jednotlivá kódová slova, kde:

- Červená značí poslední (páté) kódové slovo.
- Modrou barvou je předposlední (čtvrté) kódové slovo.
- Třetí kódové slovo má barvu zelenou.
- Oranžová barva je pro druhé kódové slovo.
- Purpurová barva značí první kódové slovo.

Ponechají-li se stejné barvy pro označení kódových slov, z diagonál je možné vytvořit následující výslednou matici D:

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{4.7}$$

Kde každý řádek odpovídá jednomu kódovému slovu.

4.8.3 Hammingovo dekódování

Na získana kódová slova je použit dekódovací proces ve funkci `hammingDecode` pro detekci, popřípadě opravu chyb(y). Uspořádaní jednotlivých bitů kódových slov lze nalézt v tabulce 4.1

CR=4/8:	d0	d1	d2	d3	p0	p1	p2	p3
CR=4/7:	d0	d1	d2	d3	p0	p1	p2	
CR=4/6:	d0	d1	d2	d3	p0	p1		
CR=4/5:	d0	d1	d2	d3	p4			

Tabulka 4.1: Uspořádaní bitů v kódových slovech [22]

Mezi paritními a datovými bity pak existují vztahy dle vzorců 4.8, 4.9, 4.10, 4.11 [22].

$$p_0 = d_0 \oplus d_1 \oplus d_2 \tag{4.8}$$

$$p_1 = d_1 \oplus d_2 \oplus d_3 \tag{4.9}$$

$$p_2 = d_0 \oplus d_1 \oplus d_3 \tag{4.10}$$

$$p_3 = d_0 \oplus d_2 \oplus d_3 \tag{4.11}$$

Jak již bylo zmíněno, v případě, že CR=4/5 počítá se sudá parita. V ostatních případech se počítá syndrom.

Je-li CR=4/6, vypočítá se p_0 dle 4.8 a p_1 dle 4.9. Pro zjištění chyby se mezi oběma výsledky použije operace OR $(p_0|p_1)$, a pokud je jedna ze zjištěných hodnot nenulová, byla detekována chyba.

Pro CR=4/7 se uvažuje, že je vypočítán syndrom s vztahem:

$$s = p0 + (p1 << 1) + (p2 << 2) \tag{4.12}$$

Pak je možné opravit jednu chybu, kdy výsledek syndromu s ukazuje na pozici, již je potřeba opravit. Znamená to tedy, že pokud:

- $s = 6$ je změněna hodnota bitu pro d_3.
- $s = 3$ je opraven bit d_2.
- $s = 7$ je zrekonstruován bit d_1.
- $s = 5$ je změněn bit d_0.

V případě, že CR=4/8 počítá se syndrom s dle vztahu:

$$s = p0 + (p1 << 1) + (p2 << 2) + (p3 << 3) \tag{4.13}$$

Potom je také možné opravit jednu chybu, kdy výsledný syndrom je o osm větší než v předchozím případě pro bity d_3, d_2 a d_0.

V obou případech platí, že pokud je výsledný syndrom nulový, tak chyba nanastala, nebo při přenosu mohlo vzniknout více chyb.

Výsledkem dekódování jsou datové nibbly ($[d_0\ d_1\ d_2\ d_3]$), kde d_0 je nejméně významný bit (LSB). Z toho vyplýva, že datovné nibbly postupují do dalšího kroku v podobě $[d_3\ d_2\ d_1\ d_0]$.

4.8.4 Dewhitener

Po opravě/detekce chyb je ve funkci `dewhiten` na nibbly aplikována tzv. `whitening` sekvence, která byla převzata z [7]. Sekvence je pole bytových hodnot o velikosti 255. Princip mechanismu spočívá (viz výpis 4.9) v aplikování XOR operace jedné bytové hodnoty ze sekvence na dva sousedící nibbly, kde první příchozí nibble je spodní a druhý příchozí horní. Oba pak tvoří jednu bytovou hodnotu, což tvoří výslednou dekódovanou hodnotu. Při příchodu dalších dvou následujících nibblů je použita následující bytová hodnota z bělící (`whitening`) sekvence a tak dále, dokud jsou k dispozici nibblové hodnoty.

```
# Apply dewhitening sequence
low = low ^ (W_SEQ[off] & LOW_NIBBLE_MASK)
```

```
high = high ^ (W_SEQ[off] & HIGH_NIBBLE_MASK) >> 4

# Construct byte
b = (high << 4 | low)
```

Výpis 4.9: Dewhitening

4.8.5 Dekódovací proces

Jakmile se data demodulují, je nejdříve nutné rozhodnout zda se jedná o zprávu s explicitní hlavičkou, a nebo s implicitní hlavičkou. Je-li použita explicitní hlavička, jsou v ní informace o velikosti užitenčých dat, jaký kódovací poměr je použit a pokud je CRC součástí zprávy. V případě použití implicitní hlavičky není ani jedna informace přítomná a za preambuli okamžitě následují užitečná data (viz kapitola 3.6.1).

Proto je ihned po demodulaci otestováno, zda je přítomna explicitní hlavička. Pro explicitní hlavičku je vždy použit CR 4/8, a tedy je pro její stanovení bráno vždy prvních 8 demodulovaných symbolů. Explicitní hlavička má také vždy o dvě hodnoty menší činitel rozprostření (SF_{head}), než jaký je použit v signálu (SF_{sig}). Platí tedy, že: $SF_{head} = SF_{sig} - 2$. Nicméně fyzicky mají symboly činitel rozprostření signálu (SF_{sig}). Proto je nutné prvních 8 symbolů posunout o dva bity doprava. Poté je na dané symboly aplikována celá dekódovací část až na aplikování bělící sekvence - ta je zcela v tomhle případě vynechána. Po dekódování je vypočteno CRC hlavičky a pokud je CRC validní - je vypočítán použitý kódovací poměr, zjištena délka užitečných dat a určeno, zda je přítomno CRC užitečných dat. Pokud nevyjde CRC hlavičky, je předpokládáno, že se jedná o zprávu s implicitní hlavičkou.

Nutno podotknout, že pokud je použita explicitní hlavička, můžou se nacházet v prvních osmi symbolech i datové byty. Pro hlavičku je vyhrazeno pouze prvních 5 bytů, a proto při přijetí signálu se SF8 a výš, jsou v prvních 8 symbolech i datové byty.

Po rozhodnutí o použité hlavičce se pro dekódování nejdříve od každé demodulované hodnoty odečte hodnota jedna. Následně se určí, zda přenos není v LDR (`Low Data Rate`) módu, který se určuje jako [7]:

$$isLDR = ((1 << sf) \cdot 1e3/bw) > LDR\_MS \tag{4.14}$$

Kde konstanta `LDR_MS` je rovna šestnácti.

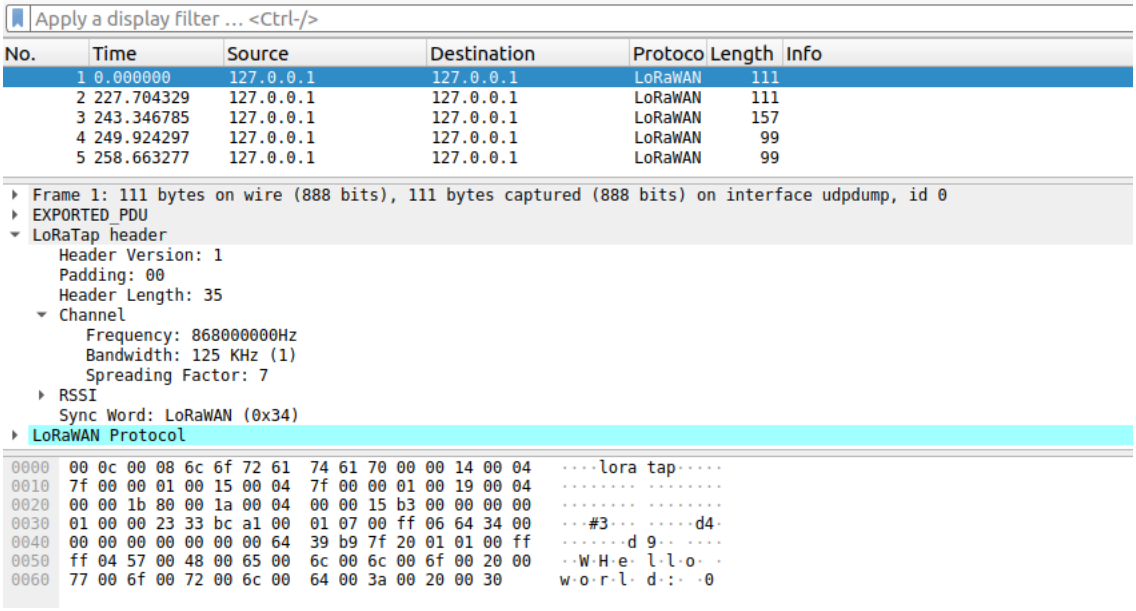
Je-li zjištěno, že je použit LDR mód, každá demodulovaná hodnota je posunuta o dva bity doprava. Obdobně jako u hlavičky i zde jsou dekódované hodnoty rozprostřeny s o dvě menším činitelem rozprostření, než jak je tomu fyzicky u symbolů. Pokud není použit, neděje se nic.

Po té je použito Grayovo kódování, inverzní operace k prokládání a Hammingovo dekódování. Před aplikováním bělící sekvence je nezbytné připojit případné datové byty z hlavičky na začátek dekódovaných bytových hodnot a až pak jí aplikovat.

Konstantou `IS_LORAWAN` lze určit, zda se zachytávají LoRaWAN zprávy, či pouze LoRa modulované zprávy. Pokud je určeno, že se zachytávají LoRaWAN zprávy, jsou dekódována data procházena a v nich se hledá adresa zařízení, sekvenční čítač a užitečná data. Pokud je nalezena adresa zařízení a pro zařízení existuje aplikační relační klíč, jsou data odšifrována a předána do softwaru Wireshark. Na druhou stranu, pokud není zachytávan LoRaWAN rámec, jsou dekódována data předána do softwaru Wireshark tak, jak jsou.

4.9 Rozhraní pro přístup k zpracovaným datům

Jako rozhraní slouží software Wireshark. V python části se vytvoří LoRaTAP hlavička s informacemi jako například použitý činitel rozprostření a šířka pásma. Data jsou převedena z decimálních hodnot do bytových a společně s vytvořenou hlavičkou jsou poslány pomocí UDP přes localhost do softwaru Wireshark. Podrobněji lze vidět zprostředkovaný výstup na obrázku 4.13.



Obrázek 4.13: Ukázka výstupu v softwaru Wireshark

Na obrázku lze vidět celkově pět poslaných zpráv. Každá má své časové razítko, uvedený použitý protokol a délku rámce. Při pokliknutí na daný rámec, lze otevřít LoRaTAP hlavičku, kde je uvedeno, jak je daná hlavička dlouhá, jakou frekvenci má přijatý rámec, jeho šířka pásma a činitel rozprostření. RSSI je vyplněno předpřipravenými hodnotami, protože není známo. Synchronní slovo (`Sync Word`) je vždy nastaveno na hexadecimální hodnot $0x34$, která se standardně používá ve většině LoRaWAN

sítích. Nicméně v budoucnu by se mohla implementovat funkce, která by dané slovo zjišťovala, kdy dané slovo by mělo být součásti preambule (dva modulované průběhy mezi up-chirp symboly a down-chirp symboly). Užitečná data lze nalézt ve spodní liště na konci. V tomhle případě se jedná o textový řetězec "Hello world: 0."

Pro spuštění softwaru wireshark jako rozhraní lze použít následující příkaz:

```
wireshark -i udpdump -k
```

Výpis 4.10: Příkaz pro spuštění softwaru wireshark

Pokud by nastala nějaká chyba, lze zkusit využít příkaz ve výpisu 4.11, který mi pomohl.

```
sudo chmod +x /usr/bin/dumpcap
```

Výpis 4.11: Příkaz pro případnou chybu se spuštěním softwaru Wireshark

4.10 Testovací prostředí

Pro testování byl využit ACR-CV modul, který mi poskytla společnost ACRIOS Systems s.r.o a LFSR algoritmus (viz výpis 4.12), jehož vstupem je osmi bitové číslo, které představuje počáteční hodnotu i délku užitečných dat. Vygenerovaná data tvoří obsah LoRa zprávy, která je poslána modulem s definovanými parametry. Lze tak otestovat několik variant podoby užitečných dat i délek o různých parametrech a sledovat, jak na ně analyzátor reaguje.

```
b = []
lfsr = start
for _ in range(start):
    bit = (lfsr ^ (lfsr >> 1) ^ (lfsr >> 3) ^ (lfsr >> 12)) & 1
    lfsr = (lfsr >> 1) | (bit << 15)
    b.append((lfsr >> 8))
```

Výpis 4.12: LFSR algoritmus pro generování zpráv

4.10.1 Zařízení pro zasílání LoRa modulovaných zpráv

Převodník RS485 pro LoRaWAN se skriptovacím rozhraním LUA je zařízení, které umožňuje připojení zařízení RS485 do sítě LoRaWAN. Toto zařízení se běžně používá v průmyslových a IoT aplikacích, kde je vyžadována bezdrátová komunikace pro vzdálené monitorování a ovládání zařízení RS485. Skriptovací rozhraní LUA umožňuje implementaci proprietárních protokolů, což z něj činí flexibilní a přizpůsobitelné řešení pro propojení se staršími systémy nebo vytváření vlastních řešení. Zařízení navíc podporuje běžný komunikační protokol Modbus, který je široce používán v aplikacích průmyslové automatizace. S vestavěným rádiovým modulem LoRaWAN a rozhraním

RS485 poskytuje převodník spolehlivé a efektivní bezdrátové řešení pro komunikaci zařízení RS485 na velké vzdálenosti.

Obrázek 4.14: ACR-CV-101L-R-D [23]

Zařízení umožňuje zasílat LoRa modulované zprávy za použití čipu SX1261. Dokumentace k API modulu lze nalézt na odkazu [24]. Pro vysílání byla použita funkce `api.p2p_tx()`, což je vlastně funkce, která umožňuje vysílat jakýkoliv LoRa signál s nastavitelnými parametry, kde parametry jsou:

```
uint8_t* data = luaL_checklstring(lua, 1, &dataSize);
uint32_t freq = luaL_checkinteger(lua, 2);
int SF = luaL_checkinteger(lua, 3);
int CR = luaL_checkinteger(lua, 4);
int bw = luaL_checkinteger(lua, 5);
int pwr = luaL_checkinteger(lua, 6);
int iqInverted = luaL_checkinteger(lua, 7);
int encrypted = luaL_checkinteger(lua, 8);
```

Výpis 4.13: Parametry funkce p2p_tx()

Příklad použití funkce lze vidět ve výpisu 4.14, kde je posílán textový řetězec "test"na frekvenci 868.5 MHz s činitelem rozprostření 7, šířkou pásma 125 kHz (1 je pro 250 kHz), výkon, a že IQ data nemají být invertovaná a zpráva nemá být šifrována.

```
api.p2p_tx("test", 868500000, 7, 1, 0, 10, 0, 0)
```

Výpis 4.14: Příklad použití funkce p2p_tx()

4.10.2 Popis testování

Vstupem užitečných dat je ACR-CV modul připojený k PC skrz sériový port. Pomocí LFSR algoritmu se vygenerují data a využitím softwaru Cutecom jsou zaslána přes sériový port do ACR-CV modulu. Ten na základě uvedených parametrů v příkazu vygeneruje LoRa zprávu a zašle ji. Modul je propojený redukcí s 30 dB atenuátorem se softwarově definovaným rádiem, které dané zprávy zachycuje a skrze USB rozhraní zasílá výstup do PC. Celé zapojení lze vidět na obrázku 4.15. V PC jsou zachycené IQ vzorky uložené do `.npy` souboru ve formátu: **SF_CR_BW_DÉLKA-DAT_exp.npy**, kde CR a BW označuje index v poli, kde `CR` je uspořádano jako - [CR 4/5, 4/6, 4/7, 4/8] a `BW` (šířka pásma) jako - [125 kHz, 250 kHz].

Obrázek 4.15: Testovací zapojení

V testovacím prostředí se následně načítají všechny uložené soubory, u kterých je vyzkoušeno, zda byly správně určeny parametry uvedené v názvu souboru. Parametr `DÉLKA-DAT` v názvu souboru se vezme jako vstup do LFSR algoritmu a jsou vygenerována totožná užitečná data, která předtím sloužila jako vstup do ACR-CV modulu. Poté se porovnává výstup z LFSR algoritmu a dekódována data získana z IQ vzorků právě testovaného souboru.

Otestované varianty lze podrobněji najít v příloze B. Nejdřív je uveden název souboru s příslušnými parametry a pak stav testování. Slovo **PASSED** za pomlčkou značí, že vše proběhlo v

pořádku. Pokud něco selhalo, je za pomlčkou uveden stav **FAILED** s uvedeným důvodem selhání.

4.10.3 Vyhodnocení otestovaných zpráv

Celkově bylo otestováno 105 různých variant zpráv pro všechny kódovací poměry, činitele rozprostření a pro dvě různé šířky pásma, kdy jednotlivé varianty lze vidět v tabulce 4.2. Celkově bylo nalezeno 38 chyb, kdy ve všech případech nastala chyba při porovnání užitečných dat. Nicméně chyby při porovnání užitečných dat se daly očekávat. Podívá-li se na to z jiného úhlu pohledu, tak v 27 případech bylo detekováno 5 <= odlišných bytových hodnot. Jen jednou nastal případ, kdy bylo více jako polovina špatně dekódováno a to při testování souboru `12_1_0_20_exp.npy`.

Spreading Factor	**Počet zpráv**	**Počet chybných zpráv**
7	29	9
8	23	8
9	17	6
10	14	10
11	15	1
12	7	4
Kódovací poměr	**Počet zpráv**	**Počet chybných zpráv**
CR 4/5	38	14
CR 4/6	27	11
CR 4/7	21	6
CR 4/8	19	7
Šířka pásma	**Počet zpráv**	**Počet chybných zpráv**
125 kHz	85	34
250 kHz	20	4

Tabulka 4.2: Celkový počet přijatých zpráv a chybně přijatých zpráv pro jednotlivé konfigurace

Při pohledu na tabulku a přílohu B platí, že nejvíc chyb při dekódování zaznamenaly zprávy se SF=10, kdy jich bylo dohromady 10 a tvořily podstatnou část přijatých zpráv pro tento činitel rozprostření. Na druhou stranu přijaté zprávy se SF=11 vykazovaly velmi přívětivé výsledky, kdy byla zaznamenána pouze jedna chybně dekódována zpráva s 4 chybně dekódovanými hodnotami. Poměrově jsou chyby u kódovací poměrů podobné, což je trošku překvapující, protože u CR=4/7 a CR=4/8 lze opravit jednu chybu, a tudíž by se dalo očekávat, že jejich počet chybně přijatých zpráv bude menší. Je možné, že se jednalo o část hodnot vedle sebe, která byla špatně určena. Příčina není známá, ale v budoucnu bude tato možnost prověřena.

Z uvedených dat lze říci, že analyzátor je funkční. Pro všechny přijaté zprávy byly korektně určeny parametry činitel rozprostření, šířka pásma a kódovací poměr. Většina zpráv byla dekódována bez chyby. V budoucnu je v plánu se zaměřit na nekorektně dekódované zprávy za účelem zrobustnění určení hodnot.

Kapitola 5

Závěr

Cíle práce byly převážně splněny, kdy v kapitole 3 byly popsány hlavní údaje k technologii LoRaWAN. V kapitole 4.1 byla provedena rešerše pro celkově pět dostupných řešení. Popis použitých nástrojů pro příjem LoRaWAN rámců, jejich demodulaci a dekódování pro všechny běžně používané (SF7-12) činitele rozprostření a kódovací poměry je zprostředkován v kapitole 4. V rámci textu byla v kapitole 2 popsána i modulace LoRa, která úzce souvisí s technologií LoRaWAN a v kapitole 4.10 přiblíženo, jak byla otestována funkčnost analyzátoru. Výstup z testování lze vidět v příloze B.

Nicméně dle zadání mělo být rozhraní pro zprostředkování dat v podobě waterfall grafu a tabulky. Místo tabulky byl zvolen software Wireshark, který je velice populární a nabízí širokou paletu funkcí. Ačkoliv waterfall graf představuje důležitou součást analyzátoru, který umožňuje vizuálně zprostředkovat přijímané spektrum, nebyl implementován. Danou problematiku jsem studoval a zjistil jsem, že vytvořit waterfall graf, který by se neustále překresloval a měl anotaci do tabulky, by bylo časově náročné. Ovšem i implementace analyzátoru byla velice náročná, a tudíž bylo upřednostěno, aby se maximalizovalo korektní demodulování a dekódování LoRa zpráv.

Práce má především tři přínosy. LoRa přenosový řetězec není moc znám. Ačkoliv již existují nástroje pro příjem LoRa/LoRaWAN zařízení a byly popsány jejich autory ve svých literaturách (viz kapitola 4.1). Neplatí, že všechny byly popsány podrobně, nebo se i některé rozchází v postupech implementace. Tahle práce poskytuje podrobný popis přenosového řetězce včetně dvou navržených algoritmů pro určení činitele rozprostření a šířky pásma.

Druhým přínosem je, že díky analyzátoru lze poměrně efektivně ladit a analyzovat LoRa/LoRaWAN zařízení v reálném provozu. Pro tyto účely se zcela odstraňuje nutnost používat kabelové připojení k zařízením, kdy některá můžou být nekomfortně umístěna. Uživatel si tak může vybrat vhodné, či komfortní místo a začít poslouchat.

Lze analyzovat všechny LoRa zprávy bez závislosti na frekvenčním plánu, což je přínos třetí. Analyzátor lze naladit na potřebnou frekvenci a poslouchat komunikaci, a tudíž není potřeba za tímhle účelem překonfigurovávat koncentrátor. Lze také i poslouchat na nestandardních frekvencích, což ne všechny koncentrátory dovolují.

Na druhou stranu analyzátor má několik menších nedostatků, které budou předmětem budoucího zkoumání. Nejstěžejnější z nich je, že momentální implementace neposkytuje kontinuální příjem, kdy výstup z SDR je zprostředkován po naplnění vyrovnávací paměti. Bylo zaznamenáno, že při velmi dlouhých zprávach bylo nemožné, nebo téměř nemožné je korektně přijmout. Na druhou stranu větší velikost vyrovnávací paměti při použitých parametrech nechtělo Adalm-Pluto přijmout, a tudíž byly velmi dlouhé zprávy z testování vynechány. V ideálním případě by bylo nejvhodnější navrhnout řešení, které by neustále přijímalo a přitom detekovalo užitečný signál.

Z důvodu absence hardwaru schopného posílat implicitní hlavičku a zprávy s šířkou pásma 500 kHz, nebyly obě varianty otestovány. Proto před odevzdáním práce byla možnost dekódovat implicitní hlavičku odstraněna. Nejedná se o velkou překážku, protože v případě implicitní hlavičky není znám kódovací poměr a velikost užitečných dat, a tudíž je třeba zkusit přijatou zprávu dekódovat pro všechny kódovací poměry a ošetřit možnost, kdy by mohlo při dekódování vlivem neznámé délky dat dojít k přetečení.

Bohužel nebyly otestovány ani downlink zprávy, respektive IQ invertované zprávy. Jelikož se jedná o důležitou součást analyzátoru, byla implementace ponechána. Protože jsou u downlink zpráv použity opačné symboly než v případě uplink zpráv, bylo aspoň otestováno, že lze pro downlink zprávy korektně určit činitel rozprostření i šířku pásma, tím, že se vytvořil komplexně sdružený průbeh z uplink zprávy.

I když byl brán důraz na výpočetní náročnost, dají se některé kroky zoptimalizovat jako například použití měkkého dekódování. V několika místech se také dá použít paralizace. Například při hledání periody by se mohli procházet všichni činitelé rozprostření zároveň, nebo při dekódování by mohly být hodnoty paralelně dekódovaný po velikostech odpovídající prokládací matici.

V budoucnu se také uvažuje vytvořit docker prostředí, čímž by se odstranila nutnost instalovat externí knihovny. Také by bylo zaručeno, že by byl analyzátor usazen vždy do stejného prostředí, a tak by byly odstraněny překážky vyplývající z použití rozdílných operačních systémů.

Bylo by rovněž patřičné navrhnout a vytvořit uživatelsky přívětivé grafické rozhraní, z kterého by se daly měnit jednotlivé parametry pro analýzu.

Analyzátor je také schopen přijímat pouze LoRa modulované zprávy. Ale v technologii LoRaWAN se používá i FSK a LR-FHSS modulace, a tudíž by mohl být analyzátor rozšířen o obě modulace.

Implementace rovněž předpokládá, že je v preambuli 8 up-chirp symbolů, respektive 8 down-chirp symbolů. I když je to standardní počet používány v LoRaWAN sítích, dá se parametr měnit. Proto by bylo vhodné navrhnout metodu, která by dokázala pracovat s větším počtem up-chirp symbolů, respektive down-chirp symbolů v preambuli.

Literatura

1. SEMTECH. *LoRa and LoRaWAN: Technical overview* [online] [cit. 2022-05-03]. Dostupné z: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/.
2. LORA ALLIANCE. *A technical overview of LoRa and LoRaWAN* [online] [cit. 2023-04-04]. Dostupné z: https://lora-alliance.org/resource_hub/what-is-lorawan/.
3. MAHJOUB, Takoua; SAID, Maymouna Ben; BOUJEMAA, Hatem. Experimental Analysis of LoRa Signal in Urban Environment. In: *2022 International Wireless Communications and Mobile Computing (IWCMC)* [online]. IEEE, 2022, s. 812–817 [cit. 2023-04-04]. ISBN 978-1-6654-6749-0. Dostupné z DOI: 10.1109/IWCMC55113.2022.9825315.
4. MILAROKOSTAS, Christos; TSOLKAS, Dimitris; PASSAS, Nikos; MERAKOS, Lazaros. A Comprehensive Study on LPWANs With a Focus on the Potential of LoRa/LoRaWAN Systems. *IEEE Communications Surveys & Tutorials*. 2023, roč. 25, č. 1, s. 825–867. ISSN 1553-877X. Dostupné z DOI: 10.1109/COMST.2022.3229846.
5. THE THINGS NETWORK. *Spreading Factors* [online] [cit. 2023-04-05]. Dostupné z: https://www.thethingsnetwork.org/docs/lorawan/spreading-factors/.
6. PROAKIS, John G.; SALEHI, Masoud. *Digital communications*. 5th ed. Boston: McGraw-Hill, 2008. ISBN 978–0–07–295716–7.
7. TAPPAREL, Joachim; AFISIADIS, Orion; MAYORAZ, Paul; BALATSOUKAS-STIMMING, Alexios; BURG, Andreas. An Open-Source LoRa Physical Layer Prototype on GNU Radio. In: *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2020, s. 1–5. ISBN 978-1-7281-5478-7. Dostupné z DOI: 10.1109/SPAWC48557.2020.9154273.
8. ROBYNS, Pieter; QUAX, Peter; LAMOTTE, Wim; THENAERS, William. A Multi-Channel Software Decoder for the LoRa Modulation Scheme. In: *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security*. SCITEPRESS - Science a Technology Publications, 2018, s. 41–51. ISBN 978-989-758-296-7. Dostupné z DOI: 10.5220/0006668400410051.

9. NEVLUD, Pavel; DVORSKÝ, Marek. *Přenos dat v komunikacích pro integrovanou výuku VUT a VŠB-TUO*. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2014. ISBN 978-80-248-3638-6.

10. LORA ALLIANCE. *RP002-1.0.0 LoRaWAN Regional Parameters* [online] [cit. 2022-12-15]. Dostupné z: `https://lora-alliance.org/resource_hub/rp002-1-0-0-lorawan-regional-parameters/`.

11. THE THINGS NETWORK. *LoRaWAN* [online] [cit. 2022-05-07]. Dostupné z: `https://www.thethingsnetwork.org/docs/lorawan/`.

12. LORA ALLIANCE. *What is LoRaWAN® Specification* [online] [cit. 2022-05-05]. Dostupné z: `https://lora-alliance.org/about-lorawan/`.

13. THE THINGS NETWORK. *Duty Cycle* [online] [cit. 2023-04-19]. Dostupné z: `https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/`.

14. LORA ALLIANCE. *LoRaWAN L2 1.0.4 Specification* [online] [cit. 2022-12-15]. Dostupné z: `https://resources.lora-alliance.org/technical-specifications/ts001-1-0-4-lorawan-l2-1-0-4-specification`.

15. ROBYNS, Pieter; QUAX, Peter; LAMOTTE, Wim; THENAERS, William. *gr-lora: An efficient LoRa decoder for GNU Radio*. Zenodo, 2017. Dostupné z DOI: `10.5281/zenodo.892174`.

16. KNIGHT, Matthew; SEEBER, Balint. Decoding LoRa: Realizing a Modern LPWAN with SDR. *Proceedings of the GNU Radio Conference*. 2016, roč. 1, č. 1. Dostupné také z: `https://pubs.gnuradio.org/index.php/grcon/article/view/8`.

17. BLUM, Josh. *LoRa modem with LimeSDR* [online] [cit. 2023-04-25]. Dostupné z: `https://myriadrf.org/news/lora-modem-limesdr/`.

18. ZLEVOR, Jan. *Využití SDR v LoRaWAN sítích*. Praha, 2021. Dostupné také z: `https://dspace.cvut.cz/handle/10467/96704`.

19. ANALOG DEVICES. *ADALM-PLUTO Evaluation Board* [online] [cit. 2023-04-19]. Dostupné z: `https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm-pluto.html`.

20. LICHTMAN, Dr. Marc. *PySDR: A Guide to SDR and DSP using Python* [online] [cit. 2022-04-18]. Dostupné z: `https://pysdr.org/content/pluto.html`.

21. IVANOV, Antoni; DANDANOV, Nikolay; CHRISTOFF, Nicole; POULKOV, Vladimir. *Modern Spectrum Sensing Techniques for Cognitive Radio Networks: Practical Implementation and Performance Evaluation* [online] [cit. 2022-10-18]. Dostupné z: `https://www.researchgate.net/publication/330599037_Modern_Spectrum_Sensing_Techniques_for_Cognitive_Radio_Networks_Practical_Implementation_and_Performance_Evaluation`.

22. TAPPAREL, Joachim. *Complete Reverse Engineering of LoRa PHY* [online] [cit. 2023-04-27]. Dostupné z: `https://www.epfl.ch/labs/tcl/resources-and-sw/lora-phy/`.

23. ACRIOS SYSTEMS S.R.O. *RS-485 IoT převodník* [online] [cit. 2023-04-30]. Dostupné z: `https://acrios.com/rs485-iot-prevodnik/`.

24. ACRIOS SYSTEMS S.R.O. *LUA API* [online] [cit. 2023-04-30]. Dostupné z: `https://wiki.acrios.com/en/acr-cv/lua-api`.

Příloha A

Elektronické přílohy

Součástí práce je elektronická příloha, která obsahuje níže uvedené soubory.

- Nejnovější verzi implementace analyzátoru:
  - Hlavní soubor (`main.py`).
  - Přijímač (`receiver.py`).
  - Soubor pro předzpracování signálu (`preprocessor.py`).
  - Demodulátor (`demodulator.py`).
  - Dekódér (`decoder.py`).
  - Rozhraní pro přístup k zpracovaným datům (`WiresharkStreamer.py`).
  - Konfigurační soubor (`config.py`).
- Soubor použitý pro testování zachycených zpráv (`test.py`).
- 6 zachycených průběhů v adresáři `/catched/test`.

Příloha B

Otestované varianty přijatých zpráv

```
Tested files:
File: 7_1_0_1_exp.npy - PASSED
File: 11_1_0_30_exp.npy - PASSED
File: 8_3_0_98_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/98] bytes
File: 7_1_0_137_exp.npy - PASSED
File: 7_2_0_38_exp.npy - PASSED
File: 8_2_0_37_exp.npy - PASSED
File: 12_3_0_10_exp.npy - PASSED
File: 7_3_0_69_exp.npy - PASSED
File: 7_1_1_44_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/44] bytes
File: 7_4_0_20_exp.npy - PASSED
File: 8_2_0_10_exp.npy - PASSED
File: 11_4_0_11_exp.npy - PASSED
File: 10_1_0_20_exp.npy - PASSED
File: 11_3_0_32_exp.npy - PASSED
File: 7_1_0_222_exp.npy - FAILED, reason: Payload - Wrong/Total: [5/222] bytes
File: 9_1_1_19_exp.npy - PASSED
File: 7_3_0_23_exp.npy - PASSED
File: 11_1_0_33_exp.npy - PASSED
File: 8_1_1_105_exp.npy - PASSED
File: 7_4_0_55_exp.npy - PASSED
File: 9_4_0_30_exp.npy - PASSED
File: 7_1_0_169_exp.npy - FAILED, reason: Payload - Wrong/Total: [2/169] bytes
File: 7_3_1_96_exp.npy - PASSED
File: 8_2_1_38_exp.npy - PASSED
File: 11_1_0_16_exp.npy - PASSED
```

```
File: 10_1_0_30_exp.npy - FAILED, reason: Payload - Wrong/Total: [2/30] bytes
File: 7_3_0_140_exp.npy - PASSED
File: 7_1_0_5_exp.npy - PASSED
File: 9_2_0_48_exp.npy - PASSED
File: 7_2_1_79_exp.npy - PASSED
File: 9_2_1_74_exp.npy - FAILED, reason: Payload - Wrong/Total: [11/74] bytes
File: 8_2_0_101_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/101] bytes
File: 8_2_1_32_exp.npy - PASSED
File: 12_3_0_12_exp.npy - FAILED, reason: Payload - Wrong/Total: [5/12] bytes
File: 12_1_0_20_exp.npy - FAILED, reason: Payload - Wrong/Total: [13/20] bytes
File: 11_2_0_10_exp.npy - PASSED
File: 8_1_0_37_exp.npy - PASSED
File: 9_2_0_85_exp.npy - FAILED, reason: Payload - Wrong/Total: [14/85] bytes
File: 7_2_0_14_exp.npy - PASSED
File: 8_1_0_6_exp.npy - PASSED
File: 11_4_0_31_exp.npy - FAILED, reason: Payload - Wrong/Total: [4/31] bytes
File: 8_2_1_65_exp.npy - PASSED
File: 8_1_0_22_exp.npy - PASSED
File: 8_4_0_49_exp.npy - PASSED
File: 9_4_1_21_exp.npy - PASSED
File: 7_2_1_49_exp.npy - PASSED
File: 10_2_0_11_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/11] bytes
File: 9_1_0_53_exp.npy - PASSED
File: 11_3_0_28_exp.npy - PASSED
File: 7_1_1_29_exp.npy - PASSED
File: 9_3_0_28_exp.npy - PASSED
File: 7_2_0_29_exp.npy - PASSED
File: 9_1_1_23_exp.npy - PASSED
File: 9_4_0_67_exp.npy - FAILED, reason: Payload - Wrong/Total: [8/67] bytes
File: 10_4_0_12_exp.npy - PASSED
File: 9_1_0_8_exp.npy - PASSED
File: 8_4_0_28_exp.npy - PASSED
File: 10_3_0_27_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/27] bytes
File: 7_3_1_99_exp.npy - PASSED
File: 7_1_1_47_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/47] bytes
File: 10_3_0_33_exp.npy - FAILED, reason: Payload - Wrong/Total: [2/33] bytes
File: 8_1_0_88_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/88] bytes
File: 11_1_0_43_exp.npy - PASSED
```

File: 7_1_0_201_exp.npy - FAILED, reason: Payload - Wrong/Total: [3/201] bytes
File: 12_2_0_13_exp.npy - FAILED, reason: Payload - Wrong/Total: [5/13] bytes
File: 10_1_0_40_exp.npy - FAILED, reason: Payload - Wrong/Total: [5/40] bytes
File: 8_2_0_26_exp.npy - PASSED
File: 9_1_0_27_exp.npy - PASSED
File: 8_4_1_35_exp.npy - PASSED
File: 8_2_0_178_exp.npy - FAILED, reason: Payload - Wrong/Total: [49/178] bytes
File: 10_4_0_40_exp.npy - FAILED, reason: Payload - Wrong/Total: [6/40] bytes
File: 7_4_1_122_exp.npy - PASSED
File: 7_3_1_31_exp.npy - PASSED
File: 9_2_0_65_exp.npy - FAILED, reason: Payload - Wrong/Total: [6/65] bytes
File: 7_1_0_92_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/92] bytes
File: 10_1_0_10_exp.npy - PASSED
File: 12_2_0_11_exp.npy - PASSED
File: 7_2_0_99_exp.npy - FAILED, reason: Payload - Wrong/Total: [2/99] bytes
File: 12_1_0_14_exp.npy - PASSED
File: 7_1_0_4_exp.npy - PASSED
File: 9_1_0_77_exp.npy - FAILED, reason: Payload - Wrong/Total: [4/77] bytes
File: 11_1_0_7_exp.npy - PASSED
File: 9_3_0_10_exp.npy - PASSED
File: 7_2_0_121_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/121] bytes
File: 10_4_0_27_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/27] bytes
File: 10_3_0_18_exp.npy - PASSED
File: 8_1_0_215_exp.npy - FAILED, reason: Payload - Wrong/Total: [54/215] bytes
File: 8_3_1_33_exp.npy - PASSED
File: 12_4_0_12_exp.npy - FAILED, reason: Payload - Wrong/Total: [6/12] bytes
File: 8_3_0_99_exp.npy - FAILED, reason: Payload - Wrong/Total: [3/99] bytes
File: 9_3_0_60_exp.npy - FAILED, reason: Payload - Wrong/Total: [5/60] bytes
File: 8_4_0_109_exp.npy - FAILED, reason: Payload - Wrong/Total: [8/109] bytes
File: 8_3_0_27_exp.npy - PASSED
File: 9_4_0_13_exp.npy - PASSED
File: 10_2_0_29_exp.npy - FAILED, reason: Payload - Wrong/Total: [2/29] bytes
File: 11_2_0_20_exp.npy - PASSED
File: 11_2_0_30_exp.npy - PASSED
File: 10_1_0_50_exp.npy - FAILED, reason: Payload - Wrong/Total: [12/50] bytes
File: 11_1_0_24_exp.npy - PASSED
File: 7_1_0_19_exp.npy - PASSED
File: 10_2_0_39_exp.npy - FAILED, reason: Payload - Wrong/Total: [5/39] bytes

```
File: 8_1_1_51_exp.npy - FAILED, reason: Payload - Wrong/Total: [1/51] bytes
File: 11_3_0_18_exp.npy - PASSED
File: 7_4_0_154_exp.npy - FAILED, reason: Payload - Wrong/Total: [4/154] bytes
File: 11_4_0_23_exp.npy - PASSED
```
