FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Implementace embedded aplikace přijímající standardizované zprávy z okolních dronů
Student:	David Horák
Vedoucí:	Ing. Lukáš Brchl
Studijní program:	Informatika
Obor / specializace:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Počet dronů a jejich obrat ze služeb roste exponenciálně; průmyslové aplikace, doručovací služby, světelné show, ale i klasické rekreační létání... Kvůli zvyšujícímu se riziku kolizí ve vzduchu a také novým případům použití v průmyslu se po celém světě zavádějí nová pravidla pro drony. Dle nařízení (EU) 2019/947 budou všechny riskantnější lety podmíněny povinnou výbavou formou systémů dálkové identifikace, což je digitální přenos identifikačních a telemetrických dat všem ostatním účastníkům letového provozu. Cílem této práce je implementovat aplikaci pro embedded HW, která bude umožňovat tyto identifikační údaje číst a dále předávat.

- Proveďte rešerši stávajících řešení a standardizace dálkové identifikace.
- Navrhněte a vyberte vhodné HW vybavení a SW technologie pro příjem identifikačních dat.
- Implementujte navržené řešení a také připravte doplňující nástroje pro otestování / vyhodnocení.
- Proveďte zhodnocení dosažených výsledků a navrhněte budoucí rozšíření.

Elektronicky schválil/a prof. Ing. Hana Kubátová, CSc. dne 21. února 2022 v Praze.

Bakalářská práce

IMPLEMENTACE EMBEDDED APLIKACE PŘIJÍMAJÍCÍ STANDARDIZOVANÉ ZPRÁVY Z OKOLNÍCH DRONŮ

David Horák

Fakulta informačních technologií
Katedra číslicového návrhu
Vedoucí: Ing. Lukáš Brchl
9. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat svému vedoucímu Ing. Lukáši Brchlovi a Ing. Tomáši Benešovi za cenné rady a připomínky. Dále bych chtěl také poděkovat své rodině a přátelům za podporu během mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 9. května 2022 .....

Abstrakt

Práce se zabývá implementací embedded aplikace pro přijímání identifikačních a lokalizačních zpráv z dronů. Práce shrnuje evropský standard pro přímou dálkovou identifikaci dronů a popisuje implementaci aplikace pro příjem těchto zpráv. Aplikace je řešena na platformě ESP32, konkrétně na SoC ESP32-C3. Řešení je v jazyce C a je napsáno jako modul pro operační systém reálného času Zephyr. Modul je schopný samostatně fungovat v rámci většího celku. Jednotlivé zprávy jsou přijímány pomocí technologií Bluetooth 4, Bluetooth 5 Long Range, Wi-Fi Beacon a Wi-Fi Aware. Přijímání Wi-Fi a Bluetooth současně není řešeno pouze pomocí jednoho SoC ESP32-C3, ale také byla architektura rozdělena mezi dvě ESP32-C3. Na jednom ESP32-C3 je provozována aplikace a jsou přijímány Wi-Fi zprávy. Druhé ESP32-C3 je použito jako Bluetooth Controller. Zařízení spolu komunikují pomocí transportního protokolu HCI UART. V rámci testování bylo zjištěno, že během přijímání Wi-Fi a Bluetooth současně na jednom zařízení dochází ke ztrátám jednotlivých zpráv. V případě rozdělené architektury se tento problém vyřešil.

Klíčová slova dálková identifikace, dron, embedded aplikace, RTOS, Zephyr RTOS, ESP32

Abstract

This thesis deals with implementation of embedded application for reception of identification and localization messages from drones. The thesis summarizes European standard for direct remote identification and descripes implementation of application for reception of these messages. The application is implemented on the ESP32 platform, specifically on the ESP32-C3 SoC. Solution is written in C and is written as a module for real-time operating system Zephyr. The module is able to function independently within larger projects. Messages are received using Bluetooth 4, Bluetooth 5 Long Range, Wi-Fi Beacon a Wi-Fi Aware technologies. Receiving of both Wi-Fi and Bluetooth was done both on single ESP32-C3 SoC and using separate Bluetooth Stack architecture. One ESP32-C3 was running application and reception of Wi-Fi messages and other was used for reception of Bluetooth Message. Both devices are communicating between themselves using HCI UART transport protocol. During the testing, it was found that when receiving Wi-Fi and Bluetooth at the same time on single SoC, there were lost messages. In the case of a separate Bluetooth Stack architecture, this problem has been solved.

Keywords remote id, drone, embedded application, RTOS, Zephyr RTOS, ESP32

Kapitola 1
Úvod

Bezpilotní letadla se pomalu stávají běžnou součástí našich životů. Stejně jako auta mají státní poznávací značku, podle které lze dohledat provozovatele vozidla, bude nutné také identifikovat bezpilotní letadla. Obdobná registrační značka by však pro tyto letouny byla nevhodná. Drony jsou často menších rozměrů a létají ve vyšších výškách a delší vzdálenosti od pilota. V takovém případě by nebylo možné dron identifikovat, protože z větší vzdálenosti není značka viditelná. Dále je vhodné aby bezpilotní letadla měli přehled o dění v jejich blízkosti, zejména kvůli zamezení kolizí s ostatními účastníky letového provozu. Z tohoto důvodu budou dle nařízení Evropské unie (EU) 2019/947 všechny riskantnější lety podmíněné vybavením dronu systémem dálkové identifikace. Vlastník bude povinen jej registrovat obdobně, jako je tomu v případě automobilů. Vlastník obdrží od národní instituce registrační číslo bezpilotního letadla a registrační číslo pilota. Pomocí bezdrátových technologií budou muset drony odesílat tyto registrační údaje společně s lokalizačními údaji pomocí bezdrátových technologií ostatním účastníkům provozu.

Tyto identifikační zprávy je poté možné číst pomocí zařízení, která podporují používané bezdrátové technologie. Lze použít například mobilní telefony, avšak valná většina nepodporuje všechny technologie, které standard definuje.

K přenosu identifikačních zpráv se využívají technologie Bluetooth 4, Bluetooth 5 Long Range, Wi-Fi Beacon a Wi-Fi Aware. Cílem práce je vytvoření zařízení pro přijímaní identifikačních zpráv, které podporuje všechny tyto technologie. Cílem teoretické části práce je seznámení se s existujícími řešeními pro příjem zpráv dálkové identifikace dronů, evropským standardem prEN 4709-002 pro přímou dálkovou identifikaci a technologiemi, které se používají pro přenos těchto zpráv. Dále zvolit vhodné hardwarové vybavení a softwarové technologie pro implementaci aplikace.

Cílem praktické části je návrh a implementace aplikace pro embedded hardware, která bude číst zprávy dálkové identifikace dle standardu ASD-STAN prEN 4709-002. Aplikace bude navržena s ohledem na možné budoucí rozšíření o podporu dalších standardů. Implementace ostatních standardů však není cílem této práce. Důležitou součástí práce je také otestování vytvořené aplikace.

Kapitola 2

Dálková identifikace

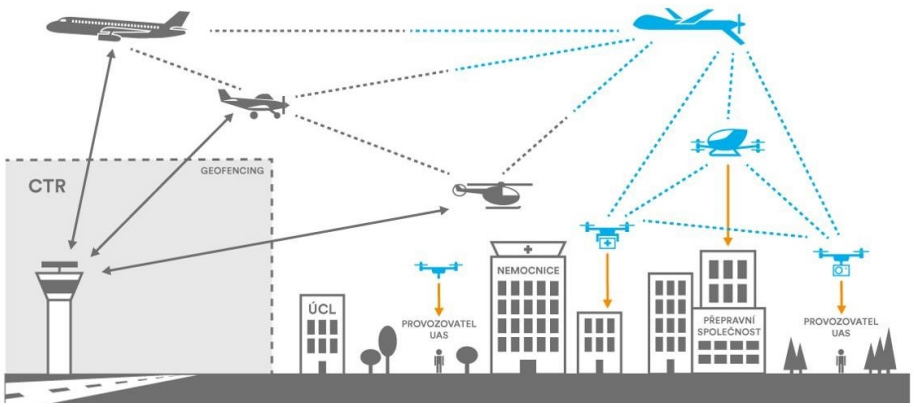
V dřívějších dobách se pro dálkovou identifikaci používal radar, který je schopný pomocí odrazu rádiových vln lokalizovat letadlo. Spolehlivost radarů je závislá na aktuálním počasí a také velikosti objektu, který chceme detekovat. V některých odvětvích letectví je radar postupně nahrazován spolehlivějšími a přesnějšími metodami dálkové identifikace.

V komerčním letectví je pro dálkovou identifikaci letadel zaveden standard Automatic Dependent Surveillance–Broadcast (ADS-B). Každé letadlo je vybaveno přístrojem vysílajícím rádiové zprávy obsahující identifikační údaje o letu. Údaje o letu jsou získávány buď pomocí globálního navigačního satelitního systému (GNSS) a nebo senzorů, jimiž je letadlo vybaveno. Tyto zprávy mohou být přijímaný ostatními účastníky provozu, nebo také letištěm. [2]

Na stejném principu funguje také dálková identifikace bezpilotních letounů. Kromě lokalizačních údajů samotného letounu jsou vysílány i informace o pilotovi. Existují dva typy dálkové identifikace:

- **Přímá dálková identifikace (DRI)** Druh dálkové identifikace při kterém letadlo do okolí vysílá lokalizační a identifikační zprávy pro ostatní účastníky letového provozu.
- **Síťová dálková identifikace (NRI)** Na rozdíl od přímé dálkové identifikace letoun nevysílá zprávy do okolí, ale lokalizační a identifikační zprávy jsou odesílány do centrálního serveru pomocí mobilní sítě. Ostatní účastníci mohou data ze serveru sledovat v reálném čase.

Evropská unie má za cíl integraci pilotovaných a bezpilotních letadel v rámci jednoho vzdušného prostoru.



Obrázek 2.1 Společný vzdušný prostor [3]

2.1 Standard ASD-STAN prEN 4709-002

Standard ASD-STAN prEN 4709-002 byl vytvořen v reakci na nařízení Evropské Unie (EU) 2019/947. Standard definuje jaké údaje se přenášejí a jakým způsobem přenos probíhá.

2.1.1 Typy odesílaných zpráv

Identifikační a lokalizační údaje se odesílají v rámci několika typů zpráv. Tyto zprávy je možné rozdělit do dvou skupin.

- **Dynamické zprávy** Do této skupiny patří zprávy, jejichž obsah se mění velmi rychle, typicky pozice a výška dronu nebo jeho rychlost. Interval odesílání těchto zpráv je dle standardu stanoven na jednu sekundu. Zároveň v době odeslání nesmí být data starší více než jednu sekundu.
- **Statické zprávy** Zprávy patřící do této skupiny obsahují hodnoty, které se v čase nemění, jako příklad je možné uvést typ letadla, ID provozovatele, či účel letu. Zprávy z této skupiny musí být odesílány aspoň jednou za tři sekundy. [4]

V tabulce 2.1 jsou zobrazeny typy jednotlivých zpráv, jejich druh, a zda je povinné je odesílat.

Tabulka 2.1 Typy zpráv dle standardu ASD-STAN prEN 4709-002 [4]

kód zprávy	typ zprávy	obsah	druh	povinné
0x0	Basic ID	identifikační informace dronu	statická	ano
0x1	Location	souřadnice, výška, rychlost, směr letu, časová značka	dynamická	ano
0x3	Self ID	účel letu, údaj vyplněný pilotem	statická	ne
0x4	System	informace o pozici pilota a skupině do které letoun patří	statická	ano
0x5	Operator ID	registrační číslo provozovatele	statická	ano
0xF	Message Pack	dovoluje seskupit více zpráv do jedné	záleží na obsahu	ano

Každá zpráva obsahuje hlavičku, která obsahuje zakódovaný typ zprávy dle tabulky 2.1. Dále je přítomna také verze protokolu. Po hlavičce následuje samotná zpráva. [4]

Tabulka 2.2 Struktura zprávy dle standardu ASD-STAN prEN 4709-002 [4]

hlavička (1 byte)		zpráva (24 bytů)
kód zprávy (4 bity)	verze protokolu (4 bity)	záleží na typu zprávy

Stejně jako není povinné odesílat všechny typy zpráv, není povinné odesílat všechny položky, které jednotlivé typy zpráv definují. Pokud některý nepovinný údaj není vysílán, neznamená to, že je údaj vynechám, ale je místo něj použita speciální hodnota pro neplatný údaj. [4]

Společně se zprávou se zároveň vysílá čítač zprávy. Každý typ zprávy má jiný čítač. Po aktualizaci dat ve zprávě vždy dojde k inkrementaci o 1. Zpráva typu Message Pack má jeden společný čítač pro všechny zprávy v ní obsažené. Čítač má délku 1 byte, může nabývat hodnot 0 až 255. Po hodnotě 255 následuje hodnota 0. [4]

2.1.1.1 Zpráva typu Basic ID

Zpráva typu Basic ID slouží k identifikace letounu. Jde o povinnou statickou zprávu. Dron je možné identifikovat pomocí následujících údajů:

- **Sériové číslo** Výrobcem přidělené sériové číslo zařízení dle formátu ANSI/CTA-2063-A-2019.
- **Registrační číslo** Číslo přidělené úřadem pro civilní letectví.
- **UTM ID** Číslo přidělené správcem vzdušného prostoru.
- **Session ID** Náhodně generované ID platné po dobu jednoho letu.

Jedna zpráva typu Basic ID může obsahovat pouze jeden identifikační údaj, pokud by bylo potřeba odesílat více údajů, je možné odesílat více zpráv typu Basic ID. Ze zmíněných údajů je povinné odesílat pouze ID v podobě sériového čísla, ostatní jsou volitelné. [4]

Tabulka 2.3 Struktura zprávy typu Basic ID [4]

délka (bytů)	položka	detaily	povinné
1	typ ID typ letounu	bity 7…4 bity 3…0	ano
20	ID letounu	ID dle uvedeného typu	ano
3	rezerva		

2.1.1.2 Zpráva typu Location

V této zprávě jsou obsaženy informace o pozici letounu.

Tabulka 2.4 Struktura zprávy typu Location [4]

délka (bytů)	položka	detaily	povinné
1	provozní stav a příznaky	provozní stav: bity 7…4 typ výšky: bit 2 V/Z segment směru: bit 1 násobič rychlosti: bit 0	ano
1	směr letu		ano
1	zemská rychlost		ano
1	vertikální rychlost		ano
4	zeměpisná šířka		ano
4	zeměpisná délka		ano
2	barometrická výška	z tlakového senzoru	ne
2	geodetická výška	z GNSS	ne
2	výška	výška letu nad zemí	ano
1	přesnost údajů	výška a souřadnice	ne
1	přesnost údajů	rychlost a barometrická výška	ne
2	časová značka		ano
1	přesnost časové značky		ne
1	rezerva		

2.1.1.3 Zpráva typu Self ID

Pomocí této zprávy může pilot dronu informovat ostatní účastníky o účelu svého letu. Není povinné vysílat zprávu Self ID.

Tabulka 2.5 Struktura zprávy typu Self ID [4]

délka (bytů)	položka	detaily	povinné
1	typ popisu		ne
23	popis	ASCII text, zbývající místo vyplněno null znaky	ne

2.1.1.4 Zpráva typu System

Zpráva typu System obsahuje údaje o pozici pilota. Ne vždy je možné získat polohu pilota z GNSS, proto se rozlišuje jakým způsobem je poloha pilota zjištěna.

- **Pozice vzletu** Pokud není řídící jednotka, pomocí které je dron ovládán, vybavena přijímačem GNSS signálu, je možné jako pozici pilota použít pozici vzletu.
- **Aktuální pozice z GNSS** Pokud je řídící jednota vybavena přijímačem GNSS signálu, dron by měl vysílat tuto pozici jako pozici pilota.
- **Fixní pozice** Další způsob, jakým je možné získat polohu pilota, pokud není řídící jednotka vybavena přijímačem GNSS. Pilot před zahájením letu nastaví pozici, kde se bude nacházet.

V případě, že je dron součástí větší skupin, měla by tato zpráva obsahovat také informace o skupině. Mezi tyto informace patří počet zařízení ve skupině a velikost oblasti, ve které létají. [4]

Tabulka 2.6 Struktura zprávy typu System [4]

délka (bytů)	položka	detaily	povinné
1	příznaky	typ klasifikace bity: 4…2 Význam pilotovy pozice: bity 1…0	ano
4	zeměpisná šířka pilota		ano
4	zeměpisná délka pilota		ano
2	velikost skupiny		ne
1	poloměr skupiny		ne
2	nejvyšší bod skupiny		ne
2	nejnižší bod skupiny		ne
1	kategorie letounů		ne
2	geodetická výška pilota		ne
5	rezerva		

2.1.1.5 Zpráva typu Operator ID

Tato zpráva obsahuje ID provozovatele letounu. ID provozovatele je přiděleno úřadem pro civilní letectví v příslušném členském státě. Provozovatel nemůže být současně registrován ve více členských státech. [4]

Tabulka 2.7 Struktura zprávy typu Operator ID [4]

délka (bytů)	položka	detaily	povinné
1	typ ID provozovatele		ano
20	ID provozovatele	Typ ID dle předchozí položky	ano
3	rezerva		

2.1.1.6 Message Pack

Pomocí této zprávy je možné zabalit více zpráv do jedné. Druh zprávy závisí na zprávách, které jsou obsaženy v Message Packu. Pokud je uvnitř Message Packu obsažena dynamická zpráva, celý Message Pack poté musí být také dynamický. V opačném případě jde o statickou zprávu. Jednotlivé zprávy jsou v souboru obsaženy včetně hlavičky. Přenos této zprávy je možný pouze u technologií, které podporují přenos většího množství dat. [4]

Tabulka 2.8 Struktura zprávy typu Message Pack [4]

délka (bytů)	položka	detaily	povinné
1	velikost jedné zprávy	vždy 25	ano
1	počet zpráv		ano
N*25	jednotlivé zprávy včetně hlavičky		ano

2.1.2 Technologie používané pro přenos

Standard ASD-STAN prEN 4709-002 definuje několik metod přenosu DRI zpráv. Jedním z požadavků při návrhu standardu bylo, aby účastník letového provozu byl schopen vidět okolní frony pomocí mobilního zařízení. Z tohoto důvodu jsou všechny metody založené na vysílání prostřednictvím technologií Wi-Fi a Bluetooth. V současné době však velká většina mobilních zařízení nepodporuje všechny technologie. Drony mají povinnost vysílat aspoň pomocí jedné z technologií. Výjimkou je Bluetooth 4, který musí být doplněn Bluetooth 5 Long Range, kvůli příliš malému dosahu. [5]

V tabluce 2.9 je srovnání bezdrátových technologií a mimo jiné je zde také srovnaná podpora mezi mobilními operačními systémy. Je překvapivé že iOS podporuje v současné době pouze Bluetooth 4, přitom na Apple zařízeních musí být také podpora pro Wi-Fi Beacon, které je potřeba pro zobrazení dostupných Wi-Fi sítí. Operační systém iOS bohužel neposkytuje vývojářům žádné rozhraní kterým je možné v aplikaci přistupovat k Wi-Fi Beacon rámcům. Na operačním systému Android je ve výchozím režimu omezená frekvence skenování Wi-Fi Beacon rámců z důvodu úspory energie, je tedy možné očekávat ztrátu přijímaných zpráv. [6], [5]

Někomu by se mohlo zdát, že udávaný dosah vysílačů je příliš velký, Wi-Fi v budově často nepokryje ani 20 m. Je třeba brát v potaz fakt, že většina letů s drony probíhá ve volném prostoru, kde signál není rušen jinými vysílači ani překážkami. Zařízení Dronetag Mini od českého startupu Dronetag dokázalo přenášet informace až do vzdálenosti 1,5 km prostřednictvím Bluetooth 5 Long Range [7].

Tabulka 2.9 Srovnání používaných bezdrátových technologií [5]

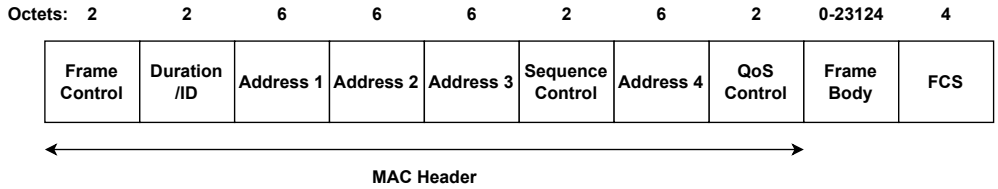
	Expected range	Maximum TX power	Android receive	Android update rate	iOS receive	iOS update rate
Bluetooth 4 Legacy Advertising	250 m	10 mW	All models	High	All models	High
Bluetooth 5 Long Range (Coded PHY S8)	1 km	10 mW	Selected newer models	High	None	None
Wi-Fi NAN	2 km	100 mW	Selected newer models	High	None	None
Wi-Fi Beacon	2 km	100 mW	All models	Low by default (see below)	Unknown	Unknown

2.1.2.1 Wi-Fi

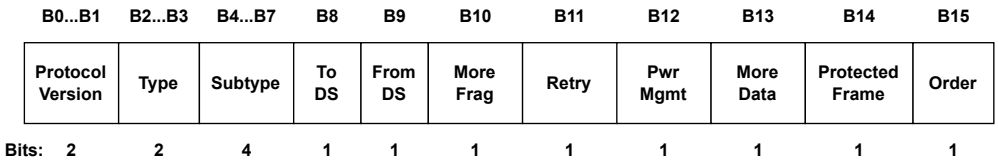
Předtím, než se budeme zabývat samotným přenosem DRI zpráv pomocí Wi-Fi, je třeba se seznámit s tím, jak vypadají data přenášená pomocí této technologie.

Standard IEEE 802.11 definuje fyzickou a linkovou vrstvu. Fyzická vrstva definuje jak se budou data fyzikálně přenášet (obecně pomocí měděného drátu, optického kabelu, elektromagnetického záření, …). Linková vrstva je zodpovědná za přístup k fyzické vrstvě a také specifikuje jak vypadají přenášená data. [8]

Základní přenášená jednotka je rámec. Rámec se skládá z hlavičky, těla a kontrolního součtu. Struktura rámce včetně položek hlavičky je zobrazena na obrázku 2.2. Samotná položka Frame Control je dále rozdělena dle obrázku 2.3.



Obrázek 2.2 Obecný rámec IEEE 802.11 [8]



Obrázek 2.3 Obsah položky Frame Control [8]

Význam adresních položek závisí na hodnotách bitů To DS a From DS. Ne všechny položky hlavičky jsou povinné, to je třeba si uvědomit při zpracovávání rámce. Například položka QoS Control je přítomná pouze pokud jde o rámec typu Data. Podobně položka Address 4, její přítomnost závisí na nastavení bitů To DS a From DS v položce Frame Control. Obsah těla rámce záleží na jeho typu. [8]

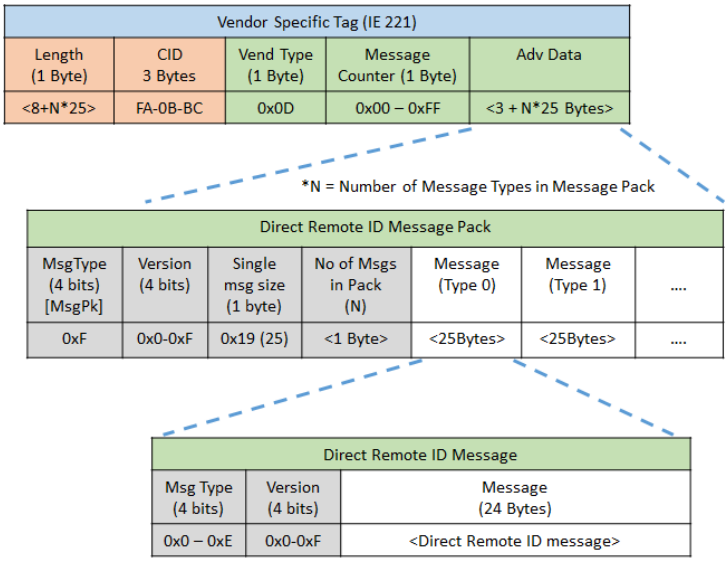
2.1.2.2 Wi-Fi Beacon

Jde o technologii, která je již dlouhou dobu využívaná. Nejrozšířenější využití má Wi-Fi Beacon pro nalezení dostupných sítí. Ještě před navázáním spojení k Wi-Fi AP je ze strany AP vysílán rámec Beacon, pomocí kterého se zařízení v okolí dozví informace o dostupné síti. Do tohoto rámce může AP vložit nějaké další informace. Právě toho je využíváno k přenosu DRI zpráv.

Beacon rámec je rámec typu Management a podtypu Beacon. Tělo Beacon rámce je rozděleno na dvě části, fixní parametry a tagované parametry. Fixní parametry obsahuji časovou značku, interval odeslání rámců a další informace o možnostech AP. Celková délka bloku obsahujícího fixní parametry činí 12 bytů. Následují tagované parametry. Každý tagovaný parametr obsahuje hlavičku. V hlavičce se nachází typ parametru a jeho délka v bytech. Mezi tagované parametry patří například SSID nebo informace o podporovaných rychlostech spojení mezi AP a koncovým zařízením. [8]

Jedním z tagovaných parametru je také Vendor Specific. Vendor Specific tag slouží k přenosu informací, které nejsou definované standardem IEEE 802.11. Hlavička má délku 5 bytů. Kromě typu tagovaného parametru a jeho délky obsahuje navíc OUI. Položka OUI slouží k unikátní identifikaci organizace. OUI je organizaci přidělené společností IEEE. Délka parametru může být od 3 do 255 bytů. Beacon rámec může obsahovat několik Vendor Specific tagů. Maximální počet je omezený pouze maximální velikostí rámce. [8]

Standard ASD-STAN využívá k přenosu DRI zpráv parametr typu Vendor Specific. Struktura tagu obsahující DRI zprávu je znázorněna na obrázku 2.4. Po hlavičce tagu následují parametry OUI (3 byty, unikátní identifikátor organizace, někdy také označováno zkratkou CID), Vend Type (1 byte), čítač DRI zprávy a samotná zakódovaná DRI zpráva. Pomocí technologie Wi-Fi Beacon je možné přenášet také zprávy typu Message Pack. [4]



Obrázek 2.4 Vendor Specific Tag obsahující DRI zprávu [4]

2.1.2.3 Wi-Fi Aware

Tato technologie umožňuje energeticky efektivní peer-to-peer komunikaci v rámci skupiny zařízení. Někdy je také označována jako Wi-Fi NAN. Finální verze specifikace byla vydána v roce 2015, avšak stále jde o velice málo rozšířenou technologii. Podpora ze strany hardware je i v současné době značně omezená. [9]

Zařízení spolu komunikují na předem určeném kanálu a v rámci určitého časového okna. Díky tomu nemusí být Wi-Fi stále aktivní a komunikace může být energeticky úspornější. Ve skupině zařízení se vždy nachází tzv. Anchor Master. Anchor Master vysílá do okolí synchronizační rámce, pomocí kterých se ostatní zařízení dozví na jakém kanálu probíhá komunikace a jak je velké časové okno, ve kterém je možné si předávat data. Časové okno začíná přijetím synchronizačního rámce a končí po určitém čase. [9]

Před samotným přenosem dat se musí zařízení zapojit do jedné synchronizované skupiny. Aby bylo možné vytvoření skupiny, zařízení si vyměňují Device Discovery rámce, díky kterým se o sobě dozvědí. Device Discovery rámce jsou vysílaný mimo časové okno komunikace, protože zařízení stále ještě nejsou synchronizovány. Následně dojde k synchronizaci zařízení pomocí synchronizačních rámců. Zařízení, která jsou schopna ostatním ve skupině poskytnout nějaká data, vysílají Service Discovery rámce. Service Discovery rámce jsou již vysílány v rámci specifikovaného časového okna. [9]

Kromě samotného přenosu dat podporuje Wi-Fi Aware také zjišťování vzdálenosti mezi jednotlivými zařízeními. Zjišťovaní vzdálenosti probíhá pomocí protokolu Fine Time Measurement (FTM).

Samotná architektura WiFi Aware je rozdělena následovně:

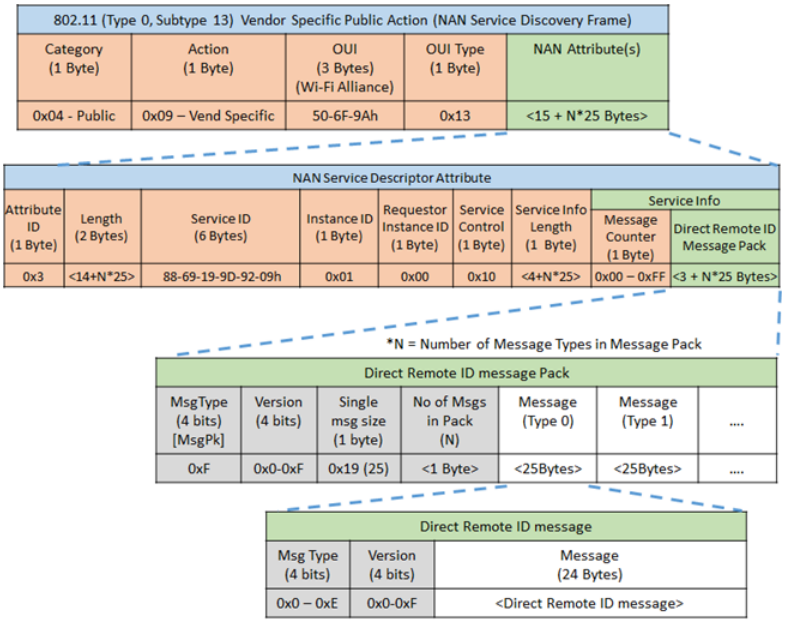
- **Discovery Engine** je zodpovědný za zpracování (případně odesílání) synchronizačních a Discovery rámců.
- **Ranging** slouží ke zjišťovaní vzdálenosti mezi zařízeními.
- **Data Engine** zajišťuje datový přenos mezi zařízeními.
- **Plánovač** je zodpovědný za správné načasování jednotlivých úkonů.
- **Linková vrstva** je oproti standardní rozšířená o další typy rámců.

Všechny zmíněné části se jako celek nazývají Wi-Fi Aware Engine. Pokud má zařízení podporu technologie Wi-Fi Aware musí implementovat všechny prvky její architektury a API, pomocí kterého je možné Wi-Fi Aware používat. [9], [10]

DRI zpráva je dle standardu ASD-STAN prEN 4709-002 přenášena pomocí rámce Service Discovery. Rámec Service Discovery je rámec typu management a podtypu action. Tělo rámce je podobně jako u Wi-Fi Beacon rozděleno na fixní parametry a atributy. Počet atributů je v rámci jednoho rámce neomezený. DRI zpráva se nachází v atributu Service Descriptor, který popisuje jakou službu dané zařízení poskytuje.

Aby bylo možné přijímat zprávy z několika dronů zároveň, je nutné aby všechna zařízení byla součástí jedné synchronizované skupiny. ID skupiny je standardem pevně stanoveno na hodnotu 50-6F-9A-01-00-FF. [4]

Pomocí Wi-Fi Aware je možné posílat DRI zprávy typu Message Pack. Vysílání probíhá na kanálu 6 ve frekvenčním pásmu 2.4 GHz nebo 149 v pásmu 5 GHz. [4]

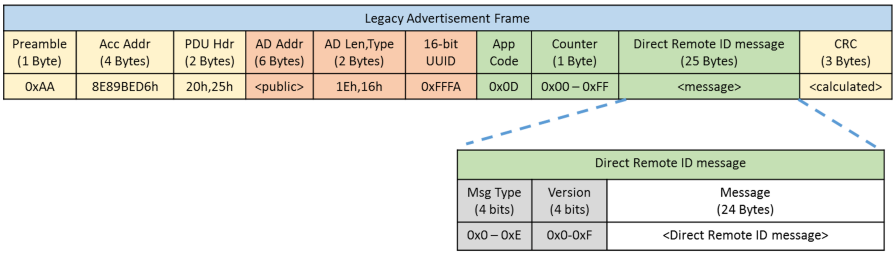


Obrázek 2.5 Rámec Service Discovery obsahující DRI zprávu [4]

2.1.2.4 Bluetooth

Technologie Bluetooth původně vznikla pro komunikace mezi dvěma zařízeními. Měla za cíl přinést alternativu pro sériovou linku RS-232. Verze 4 však přinesla řadu změn. Kromě původního, někdy nazývaného Bluetooth Classic, obsahuje nově vysokorychlostní protokol a protokol Bluetooth Low Energy (BLE). Také je již umožněno vysílání dat několika zařízením najednou. Právě protokol BLE je využíván pro přenos DRI zpráv. [11]

Bezpilotní letadlo vysílá do okolí zprávy pomocí metody advertising. Vysílání probíhá na primárních kanálech 37, 38 a 39. Data jsou postupně vysílána současně na všech třech kanálech. V rámci jednoho rámce je možné odeslat až 37 bytů dat. [4], [11]



Obrázek 2.6 Bluetooth 4 rámec obsahující DRI zprávu [4]

Verze 5 přinesla další zajímavé novinky. Bluetooth Long Range a Extended Advertising. Bluetooth Long Range významně zvyšuje dosah komunikace mezi zařízeními, specifikace slibuje až 4krát více oproti původní verzi. Toho je docíleno pomocí opravy chyb bez zpětné vazby (anglicky Forward Error Correction). Zjednodušeně zařízení vysílá o mnoho dat více, díky tomu je přijímač schopen i při větších datových ztrátách zrekonstruovat původní data. Data se stále odesílají při rychlosti 1 mb/s, avšak v závislosti na zvoleném kódovaní je reálná přenosová rychlost menší. Při zvoleném kódování S=2 je reálná přenosová rychlost 500 kb/s a při S=8 128 kb/s. Také je třeba si uvědomit, že vysílaný objem dat je mnohem větší. Při zvoleném kódování S=8 jsou vysílaná data zhruba 8krát větší oproti původním nezakódovaným datům. Pokud jsou DRI

zprávy odesílány za pomocí Bluetooth Long Range, musí být zprávy odesílány rychlostí 128 kb/s (S=8) pro dosažení největšího pokrytí. [11], [12]

Extended Advertising dovoluje odesílat větší množství dat. Oproti původním 37 bytům je možné s pomocí Extended Advertising odeslat až 255 bytů. Klasicky dochází k vysílání dat na třech advertising kanálech. Za pomocí Extended Advertising se odesílá malá hlavička na třech primárních kanálech. Hlavička odkazuje na data. Data jsou následně odeslána zvlášť na jednom z 37 datových kanálů. Je třeba zdůraznit, že podpora pro Bluetooth Long Range a Extended Advertising není povinná. Zařízení s Bluetooth 5 nemusejí tyto rozšíření podporovat. [11], [12]

2.1.3 Srovnání s ostatními standardy

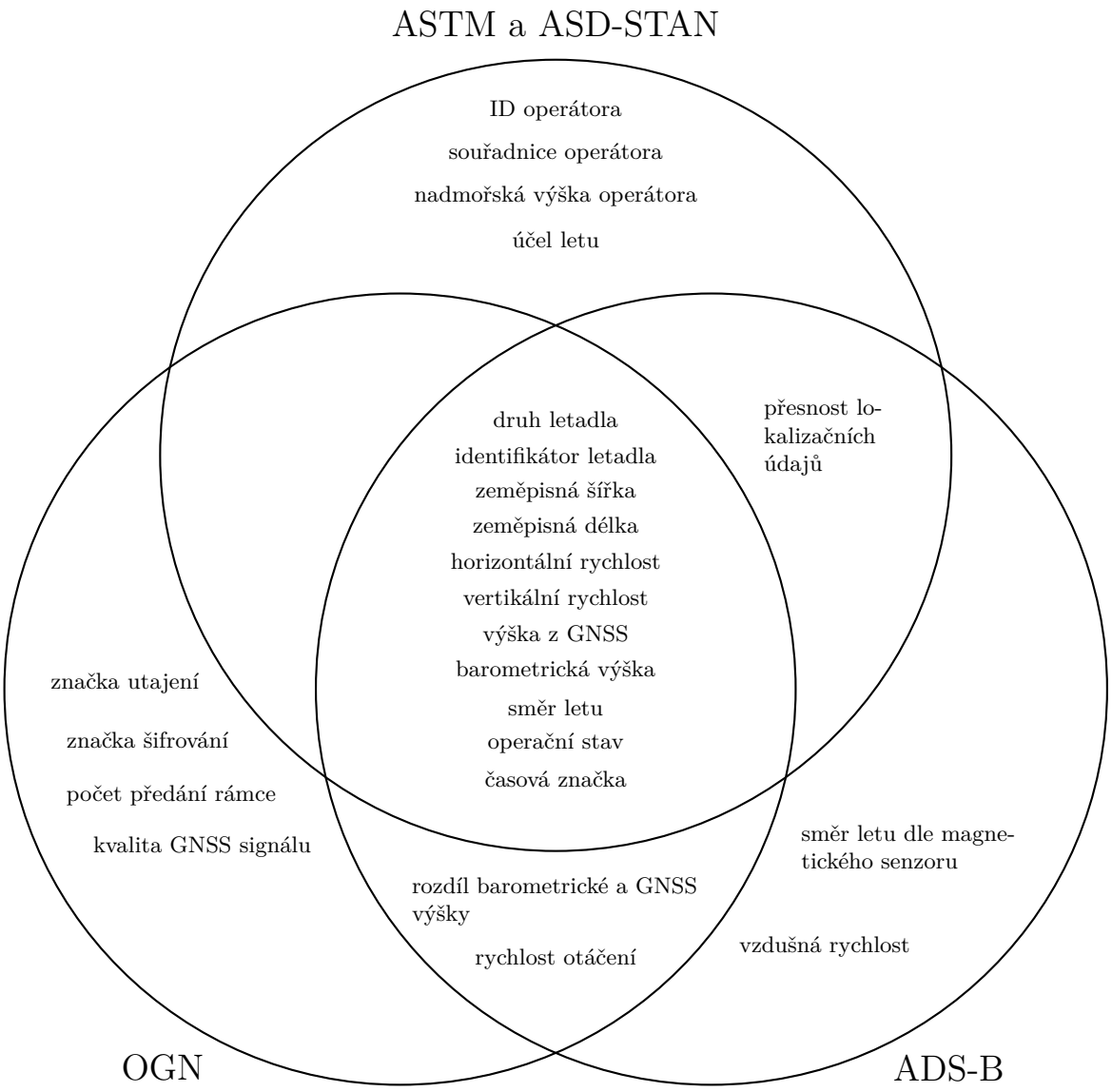
Standard ASD-STAN prEN 4709-002 reaguje na nařízení Evropské unie (EU) 2018/113, dle kterého budou muset drony létající na území evropské unie splňovat požadavky dálkové identifikace. V rámci Spojených Států Amerických existuje standard ASTM F3411, který splňuje požadavky tamní legislativy. Aby se co nejvíce minimalizovaly bariéry při implementaci přijímačů a vysílačů DRI zpráv a zajistil se určitý stupeň kompatibility na celosvětové úrovni, byl standard ASD-STAN prEN 4709-002 navržen jako kompatibilní s ASTM F3411. Jednotlivé standardy se od sebe odlišují zejména v povinnosti vysílání jednotlivých údajů. Tabulka 2.10 zobrazuje jaké typy zpráv jednotlivé standardy definují. [5]

Tabulka 2.10 Typy zpráv dle standardu ASD-STAN prEN 4709-002 a ASTM F3411 [5]

	ASD-STAN	**ASTM F3411**
Basic ID	ano	ano
Location	ano	ano
Authentication	ne	ano
Self ID	ano	ano
System	ano	ano
Operator ID	ano	ano
Message Pack	ano	ano

Pokud se nebudeme omezovat pouze na dálkovou identifikaci dronů, je možné zmínit také standard ADS-B či specifikaci Open Glider Network (OGN). Standard ADS-B je využívaný v rámci komerčních letů. Pro přenos se používají rádiové vlny o frekvenci 1090 Mhz nebo 978 Mhz. Letadlo prostřednictvím rádiových vln vysílá identifikační a lokalizační údaje získané pomocí GNSS. Informace jsou sbírány přijímači na zemi, zpravidla v blízkosti letiště. [13]

OGN je koncipován jako protokol pro komunikaci mezi větroni, drony a dalšími lehčími letouny. Mezi drony se ovšem neuchytil a nahrazují ho standardy splňující místní legislativu. Komunikace probíhá v otevřeném pásmu na frekvenci 868 MHz. [14]



Obrázek 2.7 Srovnání vysílaných informací dle vybraných standardů [4], [14], [13]

Kapitola 3

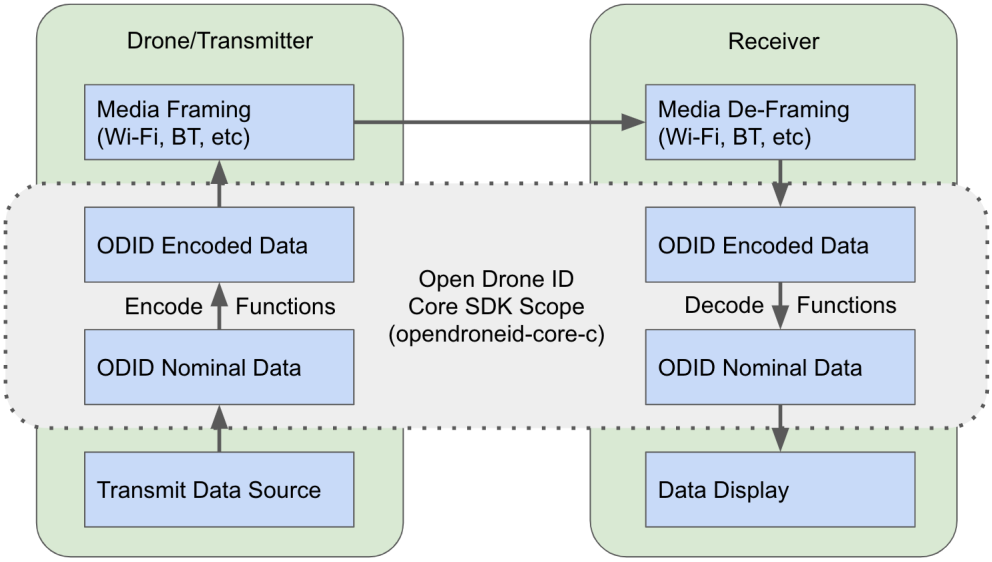
Související projekty

Tato kapitola kapitola popisuje existující projekty související s tématem dálkové identifikace bezpilotních letadel.

3.1 Open Drone ID Core C

Za pomocí této knihovny je možné snadno kódovat a dekódovat DRI zprávy. Knihovna podporuje identifikační zprávy dle standardu ASTM F3411 a ASD-STAN prEN 4709-002. Knihovna je dostupná z GitHub repozitáře [6].

Knihovna je určena pouze pro práci se zprávami přímé dálkové identifikace přenášenými pomocí Wi-Fi a Bluetooth. Není možné pracovat se zprávami síťové identifikace.

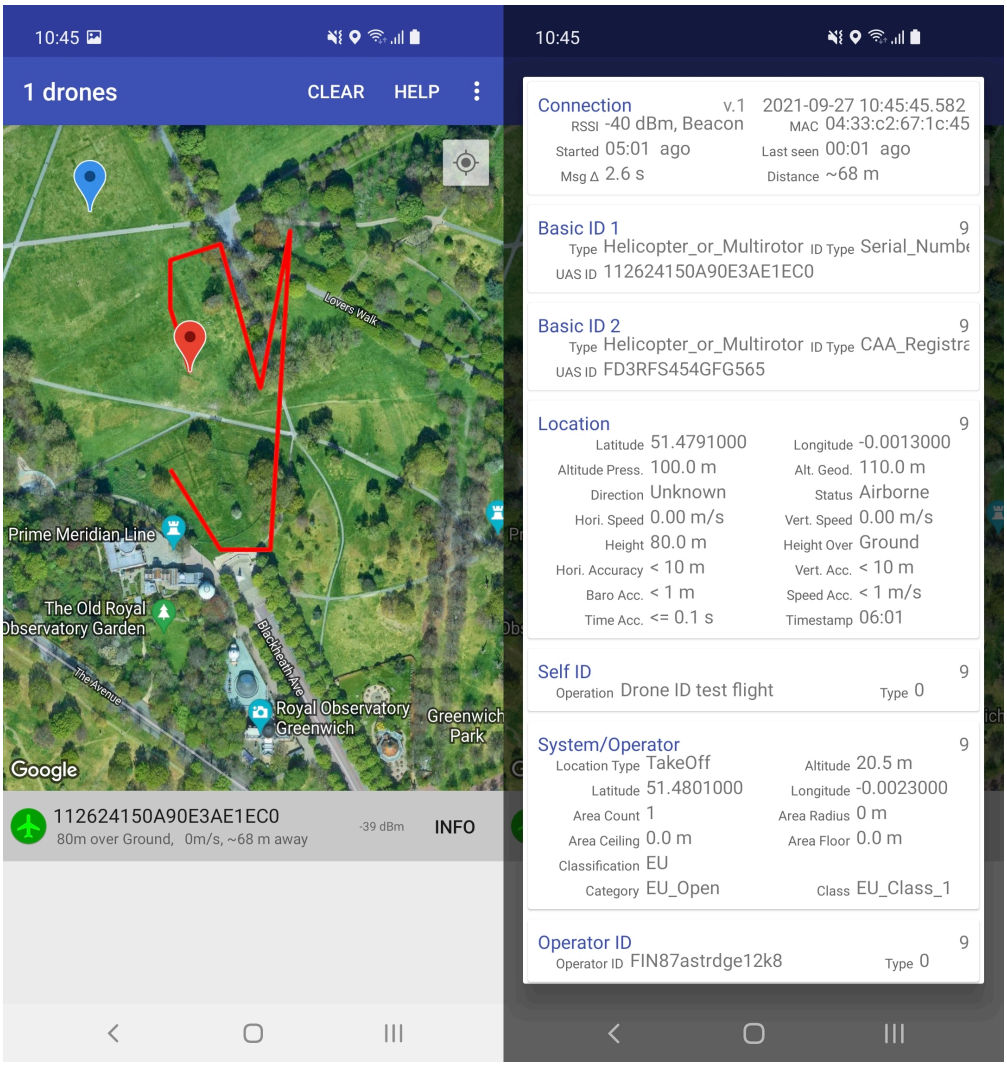


Obrázek 3.1 Architektura knihovny Open Drone ID Core C [6]

3.2 Android aplikace

Pro operační systém Android je k dispozici aplikace pro přijímání zpráv dálkové identifikace dronů. Aplikace je dostupná z GitHub repozitáře projektu [15]. Aplikace využívá knihovnu Open Drone ID Core C [6] a podporuje příjímaní zpráv prostřednictvím technologií Bluetooth 4, Bluetooth 5 Long Range, Wi-Fi Beacon a Wi-Fi Aware. Pro přijímání pomocí všech zmíněných technologií je ale také potřeba podpory ze strany mobilního telefonu. V současné době však valná většina zařízení nepodporuje všechny technologie. Nejčastěji chybí podpora pro Wi-Fi Aware a nebo Bluetooth 5 Long Range.

Po spuštění aplikace skenuje Wi-Fi a Bluetooth rámce a kontroluje zda neobsahují DRI zprávy. Pokud je zpráva nalezena, v seznamu se zobrazí viditelný letoun. Na mapě je pozice letounu znázorněna červeným špendlíkem, modrým pozice pilota nebo vzletu. Po kliknutí na položku dronu v seznamu viditelných zařízení se zobrazí okno s detailními informacemi ohledně letu.



Obrázek 3.2 Aplikace pro přijímání DRI zpráv [15]

3.3 Přijímač pro platformu ESP32

Kromě aplikace pro operační systém Android existuje také implementace přijímače pro platformu ESP32. Program je dostupný z GitHub repozitáře projektu [1]. Přijímač se nachází ve složce *id_scanner*.

Aplikace je závislá na knihovně Open Drone ID Core C. Před samotným zkompilováním a nahráním programu do zařízení ESP32 je třeba do složky s programem naklonovat repozitář s knihovnou Open Drone ID Core C. Pro kompilaci a nahrání programu do zařízení je možné použít program Arduino IDE [16]. Po nahrání programu je možné se k zařízení připojit pomocí sériové linky. Zařízení do terminálu vypisuje informace o viditelném letounu.

Ve výchozím stavu se pro příjem používá pouze Wi-Fi. Bluetooth je zatím v experimentální verzi, ale je možné ho aktivovat. Pro aktivaci Bluetooth je třeba změnit hodnotu symbolu *BLE_SCAN* v souboru *id_scanner.ino*.

```
#define DIAGNOSTICS         1
#define DUMP_ODID_FRAME     1

#define WIFI_SCAN           1
#define BLE_SCAN            1 // Experimental, does work very well.
```

Výpis kódu 3.1 Výňatek ze souboru *id_scanner.ino* přijímače pro ESP32 [1]

Přijímač je napsaný pomocí frameworku Arduino. Celá aplikace sestává pouze z jednoho zdrojového kódu jehož délka přesahuje 1000 řádku. Z tohoto důvodu jde spíše o demonstrační aplikaci, kterou nelze nijakým způsobem integrovat do většího celku.

3.4 Vysílač pro platformu ESP32

Tato práce se zabývá implementací přijímače DRI zpráv, z tohoto důvodu je vhodné mít k dispozici také protistranu v podobě vysílače, pomocí které je možné ověřit funkčnost. Vysílač pro platformu ESP32 se nachází ve složce *id_open/example* ve stejném repozitáři jako Prijímač pro ESP32. Ve složce jsou k dispozici programy *finland*, *random_flight* a *rebel_colonies*. V rámci práce byl využit pouze program *random_flight*.

Stejně jako v případě přijímače, tak i zde je třeba do složky s programem naklonovat repozitář knihovny Open Drone ID Core C. Program je také napsán pomocí frameworku Arduino, pro kompilaci a nahrání tedy lze použít Arduino IDE [16]. Po úspěšném nahrání programu je možné se k zařízení připojit pomocí sériové linky. Zařízení do terminálu vypisuje aktuální vysílanou pozici.

Pro zvolení technologií, prostřednictvím kterých má zařízení vysílat, je třeba upravit hodnoty níže uvedených symbolů v souboru *id_open.h*. Vysílač podporuje technologie Wi-Fi, Wi-Fi Aware a Bluetooth 4.

```
/*
 *  Enabling both WiFi and Bluetooth will almost certainly
 *  require a partition scheme with > 1.2M for the application.
 */

#define ID_OD_WIFI_NAN    1
#define ID_OD_WIFI_BEACON 1
#define ID_OD_BT          1        // ASTM F3411-19 / ASD-STAN 4709-002.
#define BLE_SERVICES      0        // Experimental.
```

Výpis kódu 3.2 Nastavení technologií, pomocí kterých má vysílač vysílat [1]

3.5 Vysílač pro OS Linux

Další existující implementací vysílače je program pro operační systém Linux. Projekt je dostupný ze svého GitHub repozitáře [17]. Program byl nainstalován na jednodeskový počítač Raspberry Pi 3. Podporováno je vysílání pomocí Wi-Fi Beacon a Bluetooth verze 4.

Pro vysálání zpráv pomocí Wi-Fi Beacon je třeba mít nainstalovaný program hostapd. Před zahájením programu vysílače musí být program hostapd zapnutý a nakonfigurovaný pomocí přiloženého konfiguračního souboru.

```
sudo hostapd beacon.conf
```

Výpis kódu 3.3 Spuštění programu hostapd

Následně je možné zapnout vysílání zpráv pomocí Wi-Fi Beacon.

```
sudo ./transmit b p
```

Výpis kódu 3.4 Spuštění vysílání pomocí Wi-Fi Beacon

Na zařízení Raspberry Pi bohužel nefunguje vysílání Bluetooth zpráv pomocí Extended Advertising. Pomocí technologie Bluetooth tedy není možné odesílat zprávy typu Message Pack. Spuštění vysílání zpráv pomocí technologie Bluetooth je možné pomocí příkazu:

```
sudo ./transmit l
```

Výpis kódu 3.5 Spuštění vysílání pomocí Bluetooth

Kapitola 4
Návrh

Tato kapitola popisuje zvolené softwarové vybavení a hardware pro implementaci aplikace. Dále je zde popsán návrh a architektura aplikace a návrh uživatelského rozhraní.

4.1 Zephyr RTOS

Jedním z požadavků zadavatele práce bylo, aby bylo možné vytvořenou aplikaci integrovat společně s již existujícími projekty, které jsou napsané pro Zephyr RTOS. Implementace aplikace je vytvořena jako modul do operačního systému Zephyr.

Zephyr je operační systém reálného času navržený zejména pro vestavná zařízení s omezenými zdroji a síťovou konektivitou. Jde o modulární operační systém, který je možný konfigurovat. Operační systém se skládá z jádra a dalších částí, jako jsou ovladače pro zařízení, různé knihovny a protokolové zásobníky. Operační systém obsahuje implementace protokolových zásobníků pro IPv4, IPv6, Bluetooth Low Energy. Systém je možné provozovat i na zařízeních s malou pamětí RAM. Funkčnost systému je možné omezit tak výrazně, že systém poběží i na zařízeních s 2 KB paměti RAM. [18]

RTOS Zephyr je možné konfigurovat pomocí nástrojů Kconfig a devicetree. Pomocí Kconfig je možné nakonfigurovat funkcionalitu samotného jádra operačního systému, případně jeho dalších součásti. Nástroj devicetree slouží k popisu hardwaru, na kterém Zephyr RTOS běží. Tímto způsobem je oddělena konfigurace hardwaru od samotné aplikace napsané pro tento RTOS. [18]

4.1.1 Nástroj west

Součástí projektu Zephyr [18] je mimo jiné také nástroj west. Nástroj west obsahuje sadu příkazů, pomocí kterých je možné spravovat projekt, zkompilovat a nahrát aplikaci do mikroprocesoru, v neposlední řadě je také umožněno její ladění. Příklady použití nástroje west:

```
west espressif install
```

Výpis kódu 4.1 Instalace nástrojů pro vývoj na platformě ESP32

```
west build -b esp32c3_devkitm
```

Výpis kódu 4.2 Příkaz pro kompilaci aplikace na desku ESP32-C3-DEVKITM-1

```
west flash
```

Výpis kódu 4.3 Příkaz pro nahrání aplikace do mikroprocesoru

Nástroj west umožňuje snadnou správu více projektů. Projekty jsou spravovány v rámci vlastního pracovního prostoru. Pracovní prostor je adresář obsahující podadresář *.west*, ve kterém se nachází metadata. Obsah pracovního adresáře je specifikovaný v souboru *west.yml*. Existuje několik možností jak strukturovat tento adresář. V rámci práce je použita možnost, kdy *west.yml* není součástí aplikace, ani zdrojového kódu Zephyr, ale leží ve svém adresáři *manifest*. Struktura pracovního adresáře je následující:

```
west-workspace/ ..... pracovní adresář nástroje west
├── .west/ ..... metadata nástroje west
├── dt/
│   └── opendroneid/ ..... knihovna Open Drone ID Core C
├── dri_scanner/ ..... přijímač DRI zpráv jako modul
├── scanner_application/ ..... samotná aplikace používající modul dri_scanner
├── manifest/
│   └── west.yml/ ..... popis pracovího adresáře
└── zephyr/ ..... zdrojový kód Zephyr RTOS
```

Jednotlivé projekty spravované nástrojem west musí být dostupné jako git repozitář. Samotná implementace aplikace je rozdělená do dvou částí, modulu *dri_scanner* a aplikace *scanner_application*, která s modulem pracuje. Kromě implementace je v adresáři zahrnut modul s knihovnou Open Drone ID Core C, který je k dispozici od zadavatele práce, a samotný operační systém Zephyr. Níže je zobrazen obsah souboru *west.yml*.

```
manifest:
  remotes:
    - name: dronetag
      url-base: git@bitbucket.org:dronetag
    - name: zephyr
      url-base: https://github.com/zephyrproject-rtos
    - name: horakd12
      url-base: git@bitbucket.org:horakd12
  projects:
    - name: opendroneid
      path: dt/opendroneid
      repo-path: opendroneid-core-c
      revision: v1.0-branch-dt
      remote: dronetag
    - name: dri_scanner
      path: dri_scanner
      repo-path: dri_scanner
      revision: master
      remote: horakd12
    - name: zephyr
      remote: zephyr
      repo-path: zephyr
      west-commands: scripts/west-commands.yml
      revision: zephyr-v3.0.0
      import: true
  self:
    path: manifest
```

Výpis kódu 4.4 Obsah souboru west.yml

Pro inicializaci nástroje west je třeba spustit z podadresáře *manifest* příkaz:

```
west init -l .
```

Výpis kódu 4.5 Inicializace pracovního prostoru

Tím dojde k inicializaci pracovního prostoru. Pracovní prostor zůstane prázdný dokud nedojde k jeho aktualizaci příkazem:

```
west update
```

Výpis kódu 4.6 Aktualizace pracovního pracovního prostoru

4.2 Mikroprocesor ESP32-C3

Pro příjem DRI zpráv je potřeba zařízení s podporou technologie Bluetooth 5 s rozšířením Extended Advertising a Long Range a také Wi-Fi. Komunikace s uživatelem bude probíhat pomocí textového rozhraní skrz USB či sériovou linku.

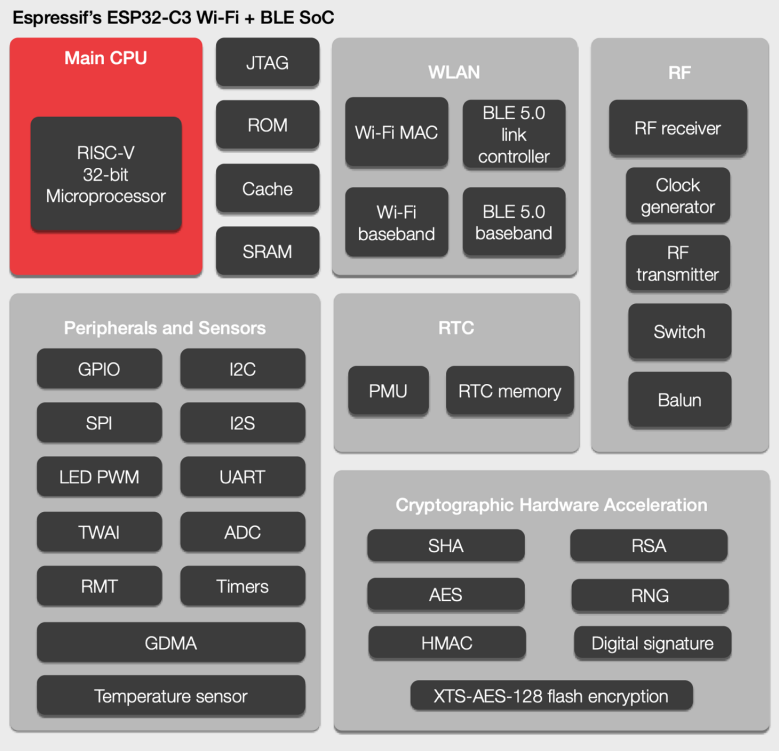
Vhodným kandidátem na použitý mikroprocesor se jeví některé zařízení platformy ESP32 od výrobce Espressif Systems. Všechna zařízení v řadě ESP32 nabízejí síťovou konektivitu pomocí technologií Wi-Fi a nebo Bluetooth.

Tabulka 4.1 Srovnání mikroprocesorů platformy ESP32 [19]

	ESP32	**ESP32-S2**	**ESP32-C3**	**ESP32-S3**
jádro	2 * Xtensa LX6, 32-bit	Xtensa LX7, 32-bit	RISC-V, 32-bit	Xtensa LX7, 32-bit
frekvence	240 MHz	240 MHz	160 MHz	240 MHz
ROM	448 KB	128 KB	384 KB	384 KB
RAM	520 KB	320 KB	400 KB	512 KB
Wi-Fi	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz
bluetooth	Bluetooth 4.2	ne	Bluetooth 5.0	Bluetooth 5.0
BT Long Range	ne	-	ano	ano
BT Extended Advertising	ne	-	ano	ano
UART	3	2	2	3
podpora Zephyr RTOS	ano	ano	ano	ne
přibližná cena za 1 kus	70 Kč	30 Kč	25 Kč	60 Kč

Žádný z mikroprocesorů nemá implementovaný Wi-Fi Aware Engine. Je ale možné Wi-Fi přepnout do tzv. promiskuitního režimu, který zachytává všechny přijaté rámce a Service Discovery rámce si dekódovat manuálně. Vzhledem k přijímání Wi-Fi Beacon rámců bude Wi-Fi poslouchat příchozí rámce neustále a není potřeba řešit synchronizaci Wi-Fi Aware s ostatními zařízeními. [10] [19]

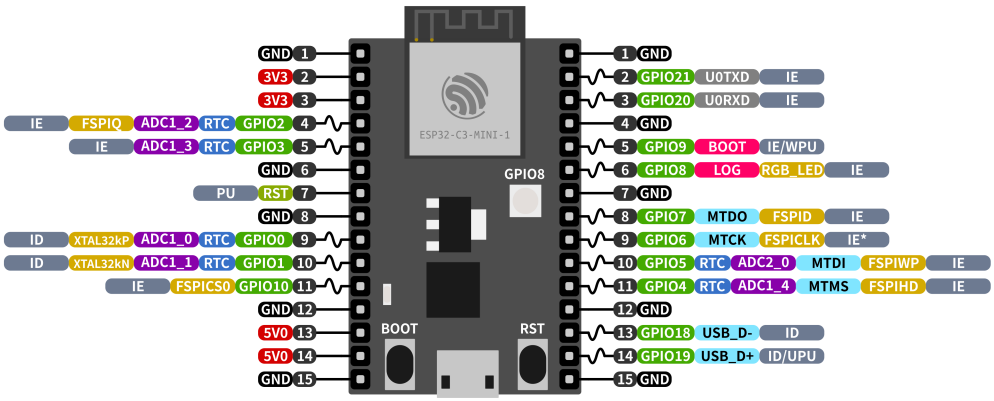
Pro vývoj aplikace byl zvolen mikroprocesor ESP32-C3. Mikroprocesor má pouze jednu sdílenou anténu, stejně jako ostatní zařízení platformy ESP32. Při současném přijímání Wi-Fi a Bluetooth je možné očekávat ztrátu zpráv. Dokumentace, v souvislosti s tímto problémem, zmiňuje pouze, že přijímání pomocí Bluetooth a Wi-Fi v promiskuitním režimu se může chovat nestabilně. Řešením může být rozdělení přijímání mezi dva mikroprocesory, kdy jeden bude sloužit k přijímání Wi-Fi a druhý Bluetooth. Je vhodné mít vyhrazenou jednu sériovou linku pro případné rozdělení Wi-Fi a Bluetooth mezi dva mikroprocesory. [20]



Obrázek 4.1 Blokový diagram mikroprocesoru ESP32-C3 [19]

4.2.1 Vývojová deska ESP32-C3-DEVKITM-1

Pro vývoj aplikace byla zvolena vývojová deska ESP32-C3-DEVKITM-1 s mikroprocesorem ESP32-C3. Má vyvedených 15 GPIO pinů, což je více než dostačující. Pro potřeby práce jsou využity pouze 4 piny pro spojení s druhou deskou, která funguje jako Bluetooth Controller, pomocí HCI UART rozhraní. Dále je na desce obsažen USB konektor pro programování desky a komunikaci prostřednictvím sériové linky. USB je připojeno k mikroprocesoru pomocí USB-UART převodníku. Dvě tlačítka BOOT a RESET slouží pro přepnutí mikroprocesoru do download režimu, respektive jeho resetování. [19]



Obrázek 4.2 Pinové vývody desky ESP32-C3-DEVKITM-1 [19]

4.3 Možnosti rozdělení Wi-Fi a Bluetooth

Existuje několik možností jak rozdělit Wi-Fi a Bluetooth rádio v rámci jednoho vestavného systému. V následujících podsekcích jsou popsány jednotlivé možnosti.

4.3.1 Wi-Fi a Bluetooth na jednom MP

Nejjednodušší je použití mikroprocesoru, který podporu obě dvě technologie. V takovém případě běží aplikace, Wi-Fi a Bluetooth komunikace na jednom mikroprocesoru. Některé mikroprocesory mají ovšem sdílenou anténu pro více bezdrátových technologií a současné využívání může přinášet problémy. V jednom čase může přistupovat k anténě pouze jedna bezdrátová technologie. Mikroprocesory toto řeší vyhrazením určitého časového okna pro jednotlivé bezdrátové technologie, například Wi-Fi je možné používat 50% času a Bluetooth také 50% času. Někdy je možné časové poměry softwarově měnit, případně si poměry mikroprocesor určuje sám podle aktuálního datového toku na jednotlivých bezdrátových technologiích. [19]

Při současném přijímání zpráv prostřednictvím Wi-Fi a Bluetooth může docházet ke ztrátám rámců, z toho důvodu je vhodnější řešit příjem jinak než pomocí sdílené antény na jednom mikroprocesoru.

Obrázek 4.3 Přijímání všech technologií na jednom mikroprocesoru

4.3.2 Separátní Bluetooth MP

Architektura Bluetooth je dělena na dvě částí, Bluetooth Host a Bluetooth Controller. Bluetooth Controller má přístup k hardwaru rádia a je zodpovědný za přenos dat. Naproti tomu Bluetooth Host komunikuje s aplikací a dává instrukce Bluetooth Controlleru. Obě části architektury mohou být přítomny na jednom mikroprocesoru, ale také je možné architekturu rozdělit mezi dva mikroprocesory. Na mikroprocesoru, kde běží aplikace je přítomen Bluetooth Host a na druhém mikroprocesoru, zajišťujícím bezdrátový přenos dat, je přítomen Bluetooth Controller. Komunikace mezi hostem a kontrolérem je zajištěna pomocí transportního protokolu HCI. Datový přenos může probíhat pomocí UART, SPI nebo USB. [11]

Příjem Wi-Fi probíhá na stejném mikroprocesoru, kde se nachází aplikace. Díky rozdělení Wi-Fi a Bluetooth přijímače je možné zároveň ve stejný čas přijímat zprávy z obou technologií.

Operační systém Zephyr je možné nakonfigurovat tak, aby byla Bluetooth architektura rozdělena mezi dvě zařízení. Po změně konfigurace není nutné nijak měnit zdrojový kód aplikace.[18]

Obrázek 4.4 Rozdělená architektura Bluetooth Stacku

4.3.3 Separátní Wi-Fi MP

Podobně jako Bluetooth je také možné oddělit mikroprocesor obsahující Wi-Fi. V tomto případě bude zařízení obsahující aplikaci obstarávat komunikaci prostřednictvím Bluetooth a také bude komunikovat s druhým mikroprocesorem zajišťujicím Wi-Fi komunikace. Zařízení se spolu mohou dorozumět například pomocí standardizovaných AT příkazů, které mohou být odesílány prostřednictvím UART protokolu. [19]

Obrázek 4.5 Oddělený mikroprocesor s Wi-Fi

4.3.4 Aplikace, Wi-Fi a Bluetooth na separátních MP

Poslední možností, jak rozdělit Wi-Fi a Bluetooth mezi více mikroprocesorů, je kombinace výše zmíněných metod. Jak pro Wi-Fi, tak pro Bluetooth komunikaci se použije separátní mikroprocesor, aplikace také poběží na separátním mikroprocesoru. Mikroprocesor s aplikací komunikuje s Wi-Fi a Bluetooth mikroprocesorem například pomocí AT příkazu, respektive HCI rozhraní.

Obrázek 4.6 Separátní mikroprocesor pro Wi-Fi a Bluetooth

4.4 Architektura aplikace

Architektura aplikace je rozdělena na několik částí, provider, služba scanner, aplikační vrstva. Jednotlivé části si mezi sebou předávají lokalizační zprávy. Lokalizační zpráva vznikne pokud provider úspěšně zachytí DRI zprávu. Zpráva se poté dostane do scanner služby, která ji dále předá aplikační vrstvě.

Lokalizační zprávy slouží pro uchovávání přijatých informací. Aplikace je navržená pro možné rozšíření o podporu dalších DRI standardů. Z tohoto důvodu je nutné řešit struktury lokalizačních zpráv pomocí polymorfizmu. Jazyk C sice není objektový, ale lze využít vnořování struktur do sebe a ukazatelů na funkce. Polymorfizmus je více popsán v kapitole Implementace.

4.4.1 Provider

Tato část aplikace je zodpovědná za příjem a dekódování přijatých lokalizačních a identifikačních zpráv. Aplikace obsahuje celkem tři providery. Jeden zajišťuje příjem Wi-Fi, tedy zajišťuje přijímání zpráv pomocí technologií Wi-Fi Beacon a Wi-Fi Aware. Druhý zajišťuje přijímaní prostřednictví Bluetooth. Posledním providerem je mock provider, který poskytuje falešné zprávy, aniž by byla potřeba fyzického přijímaní zpráv pomocí radiového signálu. Každý provider je možné aktivovat, či deaktivovat.

Pomocí providerů je zajištěn určitý stupeň hardwarové nezávislosti aplikace. V případě potřeby vyměnit mikroprocesor ESP32-C3 za nějaký jiný, stačí vytvořit nový provider, který bude umožňovat přijímaní zpráv na jiné platformě. Ostatní součásti aplikace je možné použít na všech mikroprocesorech s podporou Zephyr RTOS.

4.4.2 Služba scanner

Služba scanner sdružuje zprávy ze všech providerů a dále je může zpracovávat. Služba poskytuje filtrování zpráv na obecné úrovni a v případě potřeby je možné napsat vlastní filtr zpráv. V rámci práce je implementován filtr, který propustí pouze unikátní DRI zprávy. Po zpracování jsou zprávy předávány dále do aplikační vrstvy.

4.4.3 Aplikační vrstva

Dostává od služby scanner lokalizační zprávy a dále s nimi pracuje. Aplikační vrstva poskytuje rozhraní, pomocí kterého je možné zobrazit viditelné letouny a podrobnější informace o nich.

4.5 Uživatelské rozhraní

Aplikace uživateli poskytuje textové uživatelské rozhraní. Operační systém Zephyr obsahuje modul Shell, díky kterému lze snadno vytvářet rozhraní příkazového řádku. [18]

V základním nastavení samotný modul poskytuje příkazy pro zobrazení aktuálního stavu jádra systému (informace o vláknech, zásobnících), nastavení logování a jiné. Samozřejmě je také možné definovat své vlastní příkazy. [18]

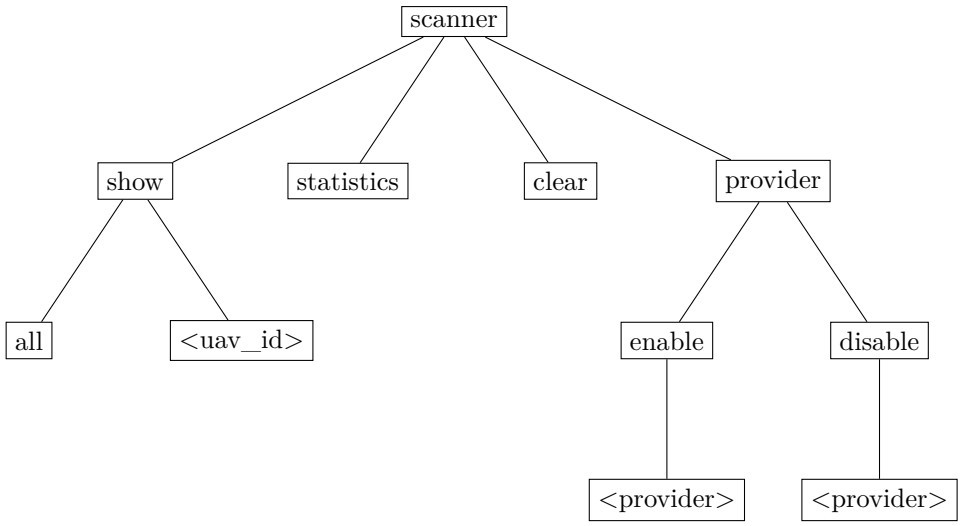
Jednotlivé příkazy jsou skupinově organizovány do stromové struktury a jsou rozděleny do následujících skupin:

- **Kořenový příkaz** Každá skupina příkazů obsahuje kořenový příkaz (nulté úrovně).
- **Statický dílčí příkaz** Dílčí příkaz (úroveň 1 a vyšší) je vždy podřazený nějakému příkazu vyšší úrovně.
- **Dynamický dílčí příkaz** Stejně jako statický dílčí příkaz, ale syntaxe příkazu je známá až za běhu programu. Může se měnit s během aplikace. [18]

Příkazy související s aplikací patří do jedné skupiny s kořenovým příkazem *scanner*. Pomocí textového rozhraní je možné vypínat a zapínat jednotlivé providery (dílčí příkaz *provider*), zobrazit statistiky související s příjmem zpráv (dílčí příkaz *statistics*) a zobrazovat informace o viditelných dronech (dílčí příkaz *show*).

Příkaz show slouží pro zobrazení informací o letounech, sám obsahuje další dílčí příkazy. *All* vypíše všechny viditelné letouny a jejich základní informace jako je sériové číslo, souřadnice, rychlost, směr letu. Podrobnější informace o letounu se zobrazí po použití příkazu s indexem letadla. Index je zobrazen během výpisu všech letounů.

Příkaz *statistics* zobrazí informace o příjmu zpráv. Je zobrazena MAC adresa vysílače, o jakou bezdrátovou technologii se jedná, celkový počet přijatých rámců, interval mezi posledními dvěma zprávami a počet ztracených zpráv. Příkazem *clear* dojde k odstranění všech zapamatovaných informací.



Obrázek 4.7 Stromová struktura příkazů textového rozhraní

Kapitola 5
Implementace

Tato kapitola obsahuje popis implementace jednotlivých částí aplikace a proces rozdělení přijímání zpráv mezi dva mikroprocesory ESP32-C3. Na závěr kapitoly je popsáno, jak je možné používat vytvořený modul.

5.1 Lokalizační zprávy

Lokalizační zpráva je struktura obsahující dekódované informace o letounu. Každá zpráva vzniká v provideru a je postupně předávána do aplikační vrstvy.

Aplikace je navržena s ohledem pro možné budoucí rozšíření o podporu dalších standardů. Obsah struktur jednotlivých lokalizačních zpráv závisí na standardu zprávy. S velkou jistotou lze konstatovat, že všechny standardy pro identifikaci letadel, jak bezpilotních, tak pilotovaných, budou obsahovat společné položky. Vždy nás bude zajímat minimálně pozice, rychlost a směr letadla. I tyto společné informace se mohou odlišovat, například různými jednotkami.

Obecná struktura lokalizační zprávy obsahuje pouze o jaký typ konkrétní implementace se jedná a obecný popis společných funkcí pro všechny implementace. Dalo by se říct, že jde o napodobení rozhraní v objektově orientovaném programování (OOP). Přístup k jednotlivým informacím zprávy je řešen pomocí funkcí. Z každé zprávy však není možné získat každou informaci. Pro příklad lze uvést zprávu typu Localization, z níž nepůjde získat identifikační údaje dronu. Z jednotlivých lokalizačních zpráv je možné získat pouze informace, které jsou v nich obsažené. Funkce vrací 0, pokud se podařilo ze zprávy získat danou informaci, a zápornou hodnotu, pokud získávání informací selže. Požadovaná hodnota je předávána pomocí výstupního parametru.

```
typedef struct localization_message localization_message;

struct localization_message_api {
    int (*size)();
    int (*get_source_id)
        (struct localization_message* msg, char* id);
    int (*get_aircraft_id)
        (struct localization_message* msg, char* id);
    int (*get_location)
        (struct localization_message* msg, float* lat, float* lon);
    int (*get_direction)
        (struct localization_message* msg, float* direction);
    int (*get_bar_altitude)
        (struct localization_message* msg, float* barAltitude);
    int (*get_geo_altitude)
        (struct localization_message* msg, float* geoAltitude);
    int (*get_horizontal_speed)
        (struct localization_message* msg, float* horizontalSpeed);
    int (*get_vertical_speed)
        (struct localization_message* msg, float* verticalSpeed);
};

struct localization_message {
    enum localization_message_type type;
    struct localization_message_api api;
};
```

Výpis kódu 5.1 Obecná definice lokalizační zprávy

Každá konkrétní implementace lokalizační zprávy musí implementovat svou vlastní strukturu a minimálně funkce definované obecným předpisem zprávy. Polymorfizmus struktur lze v jazyce C zajistit pomocí vložení obecné struktury jako první položky do struktury konkrétní implementace. Strukturu je poté možné přetypovat na obecnou i konkrétní strukturu.

Lokalizační zpráva typu DRI slouží pro uchovávání informací z DRI zpráv standardu ASD-STAN prEN 4709-002. Pro snadnější implementaci byla vyžita Open Drone ID Core C. Knihovna umožňuje dekódování zpráv do svých vlastních struktur. Lokalizační zpráva typu DRI využívá těchto struktur a obsahuje sadu funkcí pomocí kterých lze přistupovat k jednotlivým informacím. Inicializaci lokalizační zprávy typu DRI zajišťuje inicializační funkce. Funkce musí inicializovat jak konkrétní, tak i obecnou strukturu, správně nastavit typ zprávy a ukazatele na konkrétní implementace funkcí. Funkce by se dala přirovnat ke konstruktoru v OOP.

```
struct dri_localization_message {
    struct localization_message base_message;
    uint8_t mac[6];
    enum dri_wireless_technology wireless_technology;
    enum dri_standard standard;
    int16_t rssi;
    uint8_t counter;
    ODID_messagetype_t odid_type;
    ODID_UAS_Data odid_data;
};
```

Výpis kódu 5.2 Implementace lokalizační zprávy typu DRI

```
int dri_localization_message_init(struct dri_localization_message* msg){
    msg->base_message.type = LOCALIZATION_MESSAGE_TYPE_DRI;

    msg->base_message.api.size
            = &dri_localization_message_size;
    msg->base_message.api.get_source_id
            = &dri_localization_message_get_source_id;
    msg->base_message.api.get_aircraft_id
            = &dri_localization_message_get_serial_number;
    msg->base_message.api.get_location
            = &dri_localization_message_get_location;
    msg->base_message.api.get_direction
            = &dri_localization_message_get_direction;
    msg->base_message.api.get_bar_altitude
            = &dri_localization_message_get_bar_altitude;
    msg->base_message.api.get_geo_altitude
            = &dri_localization_message_get_geo_altitude;
    msg->base_message.api.get_horizontal_speed
            = &dri_localization_message_get_speed;
    msg->base_message.api.get_vertical_speed
            = &dri_localization_message_get_vertical_speed;

    memset(msg->mac, 0, 6);
    odid_initUasData(&(msg->odid_data));

    return 0;
}
```

Výpis kódu 5.3 Funkce pro inicializaci lokalizační zprávy typu DRI

5.2 Implementace providerů

Implementace jednotlivých providerů využívá model ovladače zařízení operačního systému Zephyr. Zephyr podporuje různé ovladače zařízení. Každý typ ovladače (např. UART, SPI, I2C) definuje své vlastní obecné rozhraní. Každý specifický ovladač (např. ovladač UART pro ESP32-C3) implementuje funkce obecného rozhraní. K definici ovladače slouží makro *DEVICE_DEFINE*.

```
DEVICE_DEFINE(                                  \
    dev_name, drv_name, init_fn, pm_device,     \
    data_ptr, cfg_ptr, level, prio, api_ptr     \
);
```

Výpis kódu 5.4 Makro pro definici struktury Device

Význam jednotlivých parametrů makra *DEVICE_DEFINE*:

- **dev_name** Název struktury ovladače jako C identifikátoru.
- **drv_name** Název ovladače zařízení.
- **init_fn** Ukazatel na funkci pro inicializaci ovladače.
- **pm_device** Ukazatel na strukturu pro správu napájení zařízení.
- **data_ptr** Ukazatel na strukturu s daty ovladače.
- **cfg_ptr** Ukazatel na strukturu konfigurace.

- **level** Uroveň ovladače v rámci systému (jádro/aplikace).
- **prio** Priorita inicializace ovladače.
- **api_ptr** Ukazatel na strukturu s rozhraním ovladače.

Ukazatel na ovladač je možné od systému získat pomocí jeho názvu.

```
const struct device* dev = device_get_binding("DRV_NAME");
```

Výpis kódu 5.5 Získání ukazatele na strukturu ovladače

Každý provider předává zprávy pomocí callback funkce. Existence lokalizační zprávy je garantována pouze po dobu vykonávání callback funkce. Obecné rozhraní ovladače providerů obsahuje funkce pro vypnutí, zapnutí a nastavení callback funkce určené k předání přijaté lokalizační zprávy.

```
typedef void (*message_rx_cb_t)(const struct localization_message* msg);

struct scanner_provider_api {
    int (*start)(const struct device* dev);
    int (*stop)(const struct device* dev);
    int (*set_rx_cb)(const struct device* dev, message_rx_cb_t cb);
};
```

Výpis kódu 5.6 Obecné rozhraní ovladače provider

5.2.1 Wi-Fi provider

Implementace Wi-Fi provideru vyžívá funkce specifické pro platformu ESP32 a tudíž je hardwarově závislá. Wi-Fi provider je možné používat pouze na mikroprocesorech platformy ESP32.

Mikroprocesor ESP32-C3 neobsahuje Wi-Fi Aware Engine, z toho důvodu je nutné manuálně dekódovat rámce Service Discovery. Wi-Fi lze přepnout do tzv. promiskuitního režimu, který umožňuje zpracování každého přijatého Wi-Fi rámce. Po přepnutí do promiskuitního režimu bohužel nelze používat Wi-Fi k jiným účel a je tedy nutné manuálně dekódovat také Wi-Fi Beacon rámce. Vzhledem k současnému přijímání Wi-Fi Beacon rámců bude skenování probíhat neustále, a není tedy nutné řešit synchronizaci s Wi-Fi Aware zařízeními.

Inicializační funkce Wi-Fi provideru provede inicializaci Wi-Fi mikroprocesoru a nastaví callback funkci promiskuitního režimu. Rámce Wi-Fi Beacon a SDF jsou oba typu management. Promiskuitní režim dovoluje nastavit filtr, kterým budou propouštěny pouze potřebné typy rámců.

```
esp_err_t ret = esp_wifi_set_mode(WIFI_MODE_NULL);
if(ret != ESP_OK){
  LOG_ERR("setting wifi mode failed");
  return -1;
}

ret = esp_wifi_set_promiscuous_rx_cb(&promiscious_rx_cb);
if(ret != ESP_OK){
  LOG_ERR("setting promiscuous rx callback failed");
  return -1;
}

wifi_promiscuous_filter_t filter = {
  .filter_mask = WIFI_PROMIS_FILTER_MASK_MGMT
};

ret = esp_wifi_set_promiscuous_filter(&filter);
if(ret != ESP_OK){
  LOG_ERR("setting promiscuous failed");
  return -1;
}
```

Výpis kódu 5.7 Inicializace Wi-Fi provideru

Pro dekódování Wi-Fi rámců byla vytvořena struktura *ieee80211_mgmt_frame*, na kterou se přetypuje buffer obsahující přijatý rámec. Na základě podtypu rámce lze rozhodnou zda jde o rámec Beacon či SDF. Beacon rámec je podtypu Beacon a SDF podtypu action. Typ a podtyp lze zjistit v položce *frame_control* hlavičky rámce.

```
struct ieee80211_mgmt_mac_header {
  uint16_t   frame_control;
  uint16_t   duration_id;
  uint8_t    addr1[6];
  uint8_t    addr2[6];
  uint8_t    addr3[6];
  uint16_t   sequence_control;
  //uint8_t  addr4[6];
} __attribute__((packed));

struct ieee80211_mgmt_frame {
  struct ieee80211_mgmt_mac_header header;
  uint8_t body[];
} __attribute__((packed));
```

Výpis kódu 5.8 Struktura Wi-Fi rámce

Po úspěšném nalezení DRI zprávy v rámci je zpráva dekódována pomocí knihovny Open Drone ID Core C a je z ní vytvořena lokalizační zpráva. Lokalizační zpráva je poté dále předána pomocí callback funkce.

5.2.2 Bluetooth provider

Pro implementaci Bluetooth provideru je využit Bluetooth subsystém operačního systému Zephyr. Díky tomu poběží Bluetooth provider na jakémkoliv zařízením podporujícím Bluetooth v rámci Zephyr RTOS.

Inicializace Bluetooth provideru aktivuje Bluetooth subsystém operačního systému Zephyr.

```
int err = bt_enable(NULL);
```

Výpis kódu 5.9 Blokující inicializace Bluetooth sybsystému

Po spuštění provideru dojde ke startu skenování zpráv. Pomocí struktury *bt_le_scan_param*, která je předávaná funkci pro start skenování, je možné konfigurovat parametry skenování. Existují dva typy skenování, aktivní a pasivní. Při aktivním skenování může zařízení ostatním odeslat požadavek pro skenovaní a ostatní mohou zareagovat například zkrácením intervalu odesílání zpráv. Pasivní skenovaní pouze naslouchá pro příchozí zprávy. Také je možné stanovit interval a délku časového okna. Pro zachycení co největšího množství zpráv je interval a délka časového okna nastavena na stejnou hodnotu. Ve výchozím stavu není aktivováno skenování Bluetooth Long Range rámců, je třeba aktivovat možnost *BT_LE_SCAN_OPT_CODED*.

```
struct bt_le_scan_param scanParam = {
  .type       = BT_HCI_LE_SCAN_PASSIVE,
  .interval   = 100,
  .window     = 100,
  .options    = BT_LE_SCAN_OPT_CODED
};

int err = bt_le_scan_start(&scanParam, &scan_cb);
```

Výpis kódu 5.10 Konfigurace parametrů skenování Bluetooth zpráv

Po přijetí Bluetooth zprávy zavolá Bluetooth subsystém nastavenou callback funkci. Pokud přijatá zpráva obsahuje DRI informace, dojde k dekódovaná pomocí knihovny Open Drone ID Core C a vytvoření lokalizační zpravy, pomocí callback funkce je lokalizační zpráva následně předána dále.

5.2.3 Mock provider

Mock provider umožňuje předávání falešných lokalizačních zpráv aniž by bylo nutné je fyzicky přijmout pomocí Bluetooth, či Wi-Fi. Mock provider pracuje na samostatném vlákně. Počet falešných dronů je možné konfigurovat pomocí Kconfig. Maximální možný počet je omezený pouze velikostí operační pamětí mikroprocesoru. Každý dron odesílá zprávy jednou za sekundu.

5.3 Implementace služby scanner

Služba scanner zpracovává zprávy z jednotlivých providerů a předává zprávy dále do aplikační vrstvy. Aby mohla služba scanner získávat data od provideru je nejprve nutné provider do služby zaregistrovat. Zaregistrováním provideru dojde k nastavení callback funkce pro předávání lokalizačních zpráv na interní funkci služby.

```
#include <zephyr.h>
#include <device.h>
#include <scanner_service.h>
#include <localization_message.h>

void msg_received(const struct localization_message* msg) {
  // process message
}

const struct device* providerWifi;
const struct device* providerBt;

void main(){
  providerWifi = device_get_binding("SCANNER_ESP_WIFI_PROVIDER");
  providerBt   = device_get_binding("SCANNER_BLUETOOTH_PROVIDER");

  scanner_service_init();

  scanner_service_register_provider(providerWifi);
  scanner_service_register_provider(providerBt);

  scanner_provider_start(providerWifi);
  scanner_provider_start(providerBt);

  scanner_service_set_localization_message_received_cb(&msg_received);
}
```

Výpis kódu 5.11 Příklad použití služby scanner

5.3.1 Filtrování zpráv

Služba scanner umožňuje filtraci zpráv. K jednotlivým filtrům lze přistupovat pomocí společného rozhraní. Obecná struktura filtru obsahuje ukazatele na funkce, které jsou definovány konkrétními implementacemi filtrů. Obecná struktura lze opět přirovnat k rozhraní z OOP. Funkce *can_pass* říká, zda daná lokalizační zpráva prochází filtrem nebo má být zahozena. Pomocí funkce *reset* je možné uvést filtr do výchozího stavu.

```
struct scanner_filter{
  bool (*can_pass)(
    const struct scanner_filter* filter,
    const struct localization_message* msg
  );
  int (*reset)(const struct scanner_filter* filter);
};
```

Výpis kódu 5.12 Obecná struktura filtru

V rámci práce byl implementován filtr, který propustí pouze unikátní DRI zprávy. Některá zařízení mohou stejné zprávy vysílat vícekrát. Filtr zajistí, že aplikace dostane zprávu pouze jednou. Následující výpis kódu ukazuje, jak zaregistrovat tento filtr do služby scanner:

```
const struct scanner_filter* dri_unique_filter = dri_unique_filter_get();
scanner_filter_reset(dri_unique_filter);
scanner_service_add_filter(dri_unique_filter);
```

Výpis kódu 5.13 Registrace filtru *dri_unique_filter* do služby scanner

5.4 Implementace aplikační vrstvy

Aplikační vrstva získává lokalizační zprávy od služby scanner a uchovává si získané informace o viděných dronech.

Struktura, ve které jsou uloženy informace o dronech, vyžívá polymorfizmu. Obecná struktura *aircraft* obsahuje pouze takové informace, které by měli být společné mezi zprávami různých standardů. Dále definuje předpisy pro společné funkce. Jsou zde obsaženy funkce pro aktualizaci informací o letadlech z lokalizační zprávy a naformátování informací do textového řetězce.

```
typedef struct aircraft aircraft;

struct aircraft_api {
  int (*size)();
  int (*update_from_localization_message)
      (struct aircraft* aircraft, const struct localization_message* msg);
  int (*get_short_info_string)
      (const struct aircraft* aircraft, char* str);
  int (*get_full_info_string)
      (const struct aircraft* aircraft, char* str);
  int (*get_statistics_string)
      (const struct aircraft* aircraft, char* str);
};

struct aircraft {
  enum aircraft_type type;
  struct aircraft_api api;
  char id[LOCALIZATION_MESSAGE_ID_LENGTH];
  int64_t lastSeen;
  float latitude;
  float longitude;
  float direction;
  float speed;
};
```

Výpis kódu 5.14 Obecná struktura *aircraft*

Informace o viditelných letadlech jsou uchovávány ve spojovém seznamu. K identifikaci jednotlivých letadel se používá uměle vytvořený identifikátor, pomocí kterého je možné odlišit vysílač letadla. DRI letoun není možné vždy identifikovat pomocí sériového čísla, protože sériové číslo není obsaženo ve všech typech vysílaných zpráv. MAC adresa vysílače je rozdílná pro Wi-Fi a Bluetooth. Jako umělý identifikátor pro DRI zprávu bylo zvoleno zřetězení MAC adresy s identifikátorem bezdrátové technologie. Drony, které odesílají zprávy pomocí několika technologií zároveň jsou v aplikaci viditelné několikrát, pokud by byla jako identifikátor použita pouze MAC adresa, nebylo by možné rozlišit zařízení vysílající pomocí Wi-Fi Beacon a Wi-Fi Aware současně.

Implementace spojového seznamu v operačním systému Zephyr není thread-safe, proto je nutné zajistit, aby bylo k datům spojového seznamu přistupováno v jeden čas pouze z jednoho vlákna. Toho je možné docílit pomocí mutex. Vlákno, které chce získat exkluzivní přístup k části kódu si vyžádá zamknutí mutex. Po vykonání kritické části vlákno mutex odemkne. Pokud by chtělo jiné vlákno přistupovat ke kritické části kódu, nejprve se pokusí uzamknout mutex. Pokud se vláknu nepodaří mutex uzamknout, je uspáno dokud jiné vlákno mutex neuvolní. Po probuzení se opět pokusí mutex uzamknout. [18], [21]

```
K_MUTEX_DEFINE(mutex);

void foo(){
  k_mutex_lock(&mutex, K_FOREVER);
  // critical section
  k_mutex_unlock(&mutex);
}
```

Výpis kódu 5.15 Použití mutexu v Zephyr RTOS

Po obdržení lokalizační zprávy od služby scanner dojde k vyhledání a aktualizaci struktury dronu pomocí lokalizační zprávy (funkce *update_from_localization_message()*). Pokud není struktura s letounem nalezena, vytvoří se nová. Protože je nutné zamykat mutex při přístupu ke spojovému seznamu, mohlo by docházet k blokaci vlákna obsluhující Wi-Fi a nebo Bluetooth. Jedním z možných a také použitých řešení, je vkládání přijatých zpráv do fronty a na separátním vlákně jejich čtení z fronty a zpracovávání. Před zařazením lokalizační zprávy do fronty je nutné zprávu naklonovat. Po návratu z callback funkce již není garantována jejich existence. Paměť pro struktury s informacemi o letounech a lokalizační zprávy čekající na zpracování je alokovaná ze staticky definované haldy. Velikost haldy lze konfigurovat pomocí Kconfig.

```
#include <zephyr.h>
#include <scanner_service.h>
#include <localization_message.h>

K_HEAP_DEFINE(heap, CONFIG_SCANNER_APP_HEAP_SIZE);
K_FIFO_DEFINE(fifo);

void process_msg_queue(){
  while(1){
    struct localization_message* msg = k_fifo_get(&fifo, K_FOREVER);

    // process message

    k_heap_free(&heap, (void*)msg)
  }
}

void message_received(const struct localization_message* msg){
  struct localization_message* copy = k_heap_alloc(
    &heap,
    sizeof(struct localization_message),
    K_NO_WAIT
  );

  memcpy(copy, msg, localization_message_size(msg));
  k_fifo_alloc_put(&fifo, copy);
}

K_THREAD_DEFINE(
  process_msg_queue,
  CONFIG_SCANNER_APP_THREAD_STACK_SIZE,
  check_msg,
  NULL, NULL, NULL,
  CONFIG_SCANNER_APP_THREAD_PRIORITY,
  0, 0
);
```

Výpis kódu 5.16 Způsob zpracovávání lokalizačních zpráv aplikační vrstvou

5.5 Uživatelské rozhraní

Uživatel má možnost interagovat s aplikací pomocí textového uživatelského rozhraní. Operační systém Zephyr obsahuje modul Shell pro snadnou tvorbu rozhraní příkazového řádku.

V aplikaci jsou využity pouze statické příkazy.

```
SHELL_CMD_REGISTER(syntax, subcmd, help, handler)
SHELL_CMD(syntax, subcmd, help, handler)
```

Výpis kódu 5.17 Makra pro definici statických příkazů

Makro *SHELL_CMD_REGISTER* slouží k definování kořenového příkazu. Každý dílčí příkaz nižší úrovně je možné definovat pomocí makra *SHELL_CMD*. Parametry obou maker jsou stejné.

- **syntax** Syntaxe příkazu.
- **subcmd** Odkaz na pole příkazů nižší úrovně.

- **help** Text nápovědy.
- **handler** Obslužná funkce příkazu.

Na jedné úrovni může být několik dílčích příkazů. Množinu statických dílčích příkazů je možné definovat pomocí makra *SHELL_STATIC_SUBCMD_SET_CREATE*.

```
SHELL_STATIC_SUBCMD_SET_CREATE(sub_scanner,
  SHELL_CMD(show,  &sub_show, "Show command.", cmd_show),
  SHELL_CMD(statistics,  NULL, "Show statistics.", cmd_statistics),
  SHELL_CMD(clear,  NULL, "Clear all aircrafts.", cmd_clear),
  SHELL_CMD(provider,  &sub_provider, "Provider command.", NULL),
  SHELL_SUBCMD_SET_END
);

SHELL_CMD_REGISTER(scanner, &sub_scanner, "Scanner commands", NULL);
```

Výpis kódu 5.18 Konfigurace zařízení jako Bluetooth Controlleru s aktivovaným HCI UART rozhraním

Při použití příkazu je zavolána obslužná funkce. Obslužnou funkci je možné definovat jakémukoliv příkazu na jakékoliv úrovni. Funkce je volána vždy i pro všechny nadřazené příkazy. Pokud uživatel použije příkaz *scanner clear*, nejprve bude zavolána obslužná funkce pro příkaz *scanner* a následně pro příkaz *clear*.

```
int cmd_handler(const struct shell *shell, size_t argc, char **argv){
  shell_print(shell, "hello from cmd handler")
  return 0;
}
```

Výpis kódu 5.19 Příklad obslužné funkce příkazu

Parametr *argc* obsahuje počet parametrů příkazu. Jednotlivé parametry jsou uloženy v poli *argv*. První položkou pole je název samotného příkazu.

```
uart:~$ scanner show all
1. 112624150A90E3AE1EC0
    location: [51.4790993, -0.0013000]   heading:  n/a   speed:  0.0 m/s
    Last seen: 738 ms ago
2. 7178976934099028269
    location: [50.1046600, 14.3891296]   heading: 17.0°  speed: 20.0 m/s
    Last seen: 884 ms ago
3. 3925125672748463271
    location: [50.1049881, 14.3902998]   heading: 40.0°  speed: 20.0 m/s
    Last seen: 693 ms ago
```

Výpis kódu 5.20 Ukázka příkazu pro zobrazení seznamu viditelných dronů

```
uart:~$ scanner show 1
serial number: 112624150A90E3AE1EC0
operational status: airborne
location: [51.4790993, -0.0013000]
heading: n/a
height: n/a
speed: 0.0 m/s

operator ID: FIN87astrdge12k8
opeartor location: [51.4800987, -0.0023000] (takeoff)
opeartor altitude: 20.5 m

last seen: 125 ms ago

mac: 39:FD:97:56:A2:D1
rssi: -76 dBm (Bluetooth)
received msgs: 52
lost msgs: 0

Counters:
 - Basic ID: 11
 - Location: 5
 - Auth: 17
 - Self ID: 5
 - System: 4
 - Operator ID: 4
```

Výpis kódu 5.21 Ukázka příkazu pro zobrazení informací o dronu

```
uart:~$ scanner statistics
1. 78:E3:6D:09:FD:48
    WiFi NAN     rssi: -59 dBm   msgs rx: 62     msgs lost: 0
    msg loss rate: 0.0% packet rx: 62   interval: 800 ms
2. 78:E3:6D:09:FD:48
    WiFi Beacon  rssi: -58 dBm   msgs rx: 62     msgs lost: 0
    msg loss rate: 0.0% packet rx: 62   interval: 800 ms
3. 78:E3:6D:09:FD:4A
    Bluetooth    rssi: -62 dBm   msgs rx: 157    msgs lost: 0
    msg loss rate: 0.0 % packet rx: 2206 interval: 48 ms
```

Výpis kódu 5.22 Ukázka příkazu pro zobrazení statistik příjmu zpráv

5.6 Rozdělení příjímání zpráv mezi dva mikroprocesory

Pro řešení problémů, které jsou způsobeny koexistencí více bezdrátových technologií, je příjem Wi-Fi a Bluetooth rozdělen mezi dva mikroprocesory ESP32-C3. Byla zvolena možnost kdy aplikace a příjem pomocí Wi-Fi běží na jednom ESP32-C3 a druhé funguje jako přijímač Bluetooth. Zařízení spolu komunikují pomocí transportního protokolu HCI UART.

Operační systém umožňuje bohatou konfiguraci Bluetooth subsystému, díky čemuž je možné aktivovat pouze potřebné součásti. Zařízení fungující jako Bluetooth Host vyžaduje aktivovaný Bluetooth subsystém a rozhraní HCI. Protože je využit externí Bluetooth Controller, je nutné deaktivovat součást Bluetooth Controller. Na straně Bluetooth Controlleru je třeba aktivovat Bluetooth subsystém a povolit rozhraní HCI. Níže jsou uvedeny části konfiguračních souborů *prj.conf* pro Bluetooth Host a Controller.

```
CONFIG_BT=y
CONFIG_BT_HCI=y
CONFIG_BT_CTLR=n
CONFIG_BT_H4=y
```

Výpis kódu 5.23 Konfigurace zařízení jako Bluetooth Host používající HCI UART rozhraní pro komunikaci s Bluetooth Controller

```
CONFIG_BT=y
CONFIG_BT_HCI=y
CONFIG_BT_HCI_RAW=y
CONFIG_BT_CTLR=y
CONFIG_BT_HCI_RAW=y
CONFIG_BT_HCI_RAW_H4=y
CONFIG_BT_HCI_RAW_H4_ENABLE=y
```

Výpis kódu 5.24 Konfigurace zařízení jako Bluetooth Controller s aktivovaným HCI UART rozhraním

Operační systém Zephyr obsahuje vzorový projekt *hci_uart*. Po nahrání projektu se bude zařízení chovat jako Bluetooth Controller. V projektu je obsažena hardwarová konfigurace pro velké množství zařízení, avšak pro ESP32-C3-DEVKITM-1 konfigurace chybí. V podadresáři *boards* projektu *hci_uart* se nachází overlay soubory, pomocí kterých je možné upravit výchozí hardwarovou konfiguraci zařízení. Pro konfiguraci zařízení ESP32-C3-DEVKITM-1 je třeba vytvořit soubor *esp32c3_devkitm.overlay*. V souboru HW konfigurace je aktivována druhá sériová linka *uart1* a je nastavena pro použití HCI rozhraním. Obsah souboru je následující:

```
/ {
  chosen {
        zephyr,bt-c2h-uart = &uart1;
  };
};

&uart1 {
  status = "okay";
  current-speed = <921600>;
  tx-pin = <6>;
  rx-pin = <7>;
  rts-pin = <8>;
  cts-pin = <9>;
  hw-flow-control;
};
```

Výpis kódu 5.25 Obsah souboru *esp32c3_devkitm.overlay* pro Bluetooth Controller

Podobně, pro zařízení fungující jako Bluetooth Host, je také potřeba provést konfiguraci hardwaru. Opět je třeba vytvořit soubor *esp32c3_devkitm.overlay* v podadresáři *boards* a nakonfigurovat sériovou linku *uart1* jako Bluetooth rozhraní.

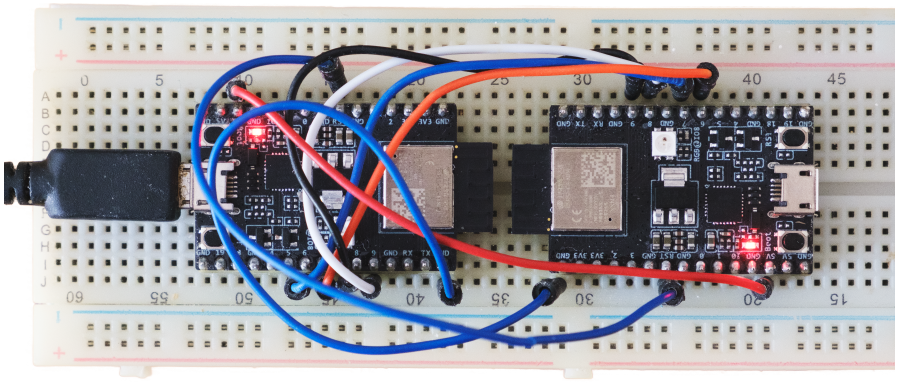
```
/ {
  chosen {
    zephyr,bt-uart = &uart1;
  };
};

&uart1 {
  status = "okay";
  current-speed = <921600>;
  tx-pin = <6>;
  rx-pin = <7>;
  rts-pin = <8>;
  cts-pin = <9>;
  hw-flow-control;
};
```

Výpis kódu 5.26 Obsah souboru *esp32c3_devkitm.overlay* pro Bluetooth Host

Komunikace pomocí HCI UART využívá mechanizmus HW Control Flow, který vyžaduje připojení pomocí čtyř vodičů, pokud počítáme vodiče čistě pro UART rozhraní. Dvě zařízení jsou spolu zapojena následovně:

Tabulka 5.1 Zapojení BT Host a BT Controller

BT Host	<–>	BT Controller
GPIO 6 (TX)	<–>	GPIO 7 (RX)
GPIO 7 (RX)	<–>	GPIO 6 (TX)
GPIO 8 (RTS)	<–>	GPIO 9 (CTS)
GPIO 9 (CTS)	<–>	GPIO 8 (RTS)
5V	<–>	5V
GND	<–>	GND
RST	<–>	RST



Obrázek 5.1 Zapojení dvou ESP32-C3 na nepájivém poli

5.7 Modul do Zephyr RTOS

Moduly operačnímu systému dovolují používat externí projekty. Sám operační systém Zephyr závisí na některých modulech, například kryptografických knihovnách nebo HAL od výrobce mikroprocesoru. Implementace aplikace je také vytvořena jako modul operačního systému. Struktura modulu je následující:

```
dri_scanner ..... modul pro Zephyr RTOS
├── scanner_app ..... aplikační vrstva
│   ├── CMakeLists.txt
│   └── Kconfig
├── scanner_service ..... služba scanner
│   ├── providers ..... provideři služby
│   │   ├── bluetooth
│   │   ├── esp_wifi_provider
│   │   ├── mock_provider
│   │   └── Kconfig
│   ├── scanner_service ..... samotná služba scanner
│   │   └── Kconfig
│   ├── CMakeLists.txt
│   └── Kconfig
├── tests ..... jednotkové testy
├── zephyr
│   └── module.yml ..... popis modulu
├── CMakeLists.txt ..... instrukce pro build systém
└── Kconfig ..... konfigurace modulu
```

Soubor *module.yml* uvnitř adresáře *zephyr* obsahuje popis modulu. Zde je specifikován název modulu, a kde se nachází soubory *Kconfig* a *CMakeLists.txt*. Modul může mít závislosti na jiných modulech. Vytvořený modul *dri_scanner* závisí na modulu s knihovou Open Drone ID Core C.

```
name: dri_scanner
build:
  cmake: .
  kconfig: Kconfig
  depends:
    - opendroneid
```

Výpis kódu 5.27 Obsah souboru *module.yml*

Ke konfiguraci modulu slouží nástroj Kconfig. V souborech *Kconfig* jsou definované konfigurovatelné symboly. Konfigurace je pro přehlednost rozdělena do několika souborů. Hlavní konfigurační soubor se odkazuje na soubory definující konfiguraci jednotlivých částí modulu.

```
menu "Scanner App"
  depends on SCANNER_SERVICE

  config SCANNER_APP
    bool "scanner application"
    depends on SCANNER_SERVICE
    default n
    help
      Enable scanner application.

  config SCANNER_APP_HEAP_SIZE
    int "scanner app heap size"
    default 32768
    depends on SCANNER_APP && SCANNER_SERVICE
    help
      Heap size of scanner app used for
      message fifo and aircraft structures.
endmenu
```

Výpis kódu 5.28 Část definice konfigurace aplikační vrstvy

Definované symboly je poté možné konfigurovat. Prvním způsobem je psát konfiguraci do souboru konfigurace projektu *prj.conf*.

```
CONFIG_SCANNER_SERVICE=y                    # povoleni sluzby scanner

CONFIG_SCANNER_APP=y                        # povoleni aplikacni vrstvy
CONFIG_SCANNER_APP_THREAD_PRIORITY=10       # priorita zpracováni
CONFIG_SCANNER_APP_HEAP_SIZE=32768          # velikost haldy aplikacni vrstvy
CONFIG_SCANNER_APP_SHELL=y                  # povoleni textového rozhrani

CONFIG_SCANNER_PROVIDER_BLUETOOTH=y         # povoleni bluetooth provideru
CONFIG_SCANNER_PROVIDER_ESP_WIFI=y          # povoleni bluetooth provideru
CONFIG_SCANNER_PROVIDER_MOCK=y              # povoleni mock provideru
CONFIG_SCANNER_PROVIDER_MOCK_PRIORITY=20    # priorita vlakna mock provideru
CONFIG_SCANNER_PROVIDER_MOCK_DATA_COUNT=5   # pocet falesnych letounu
```

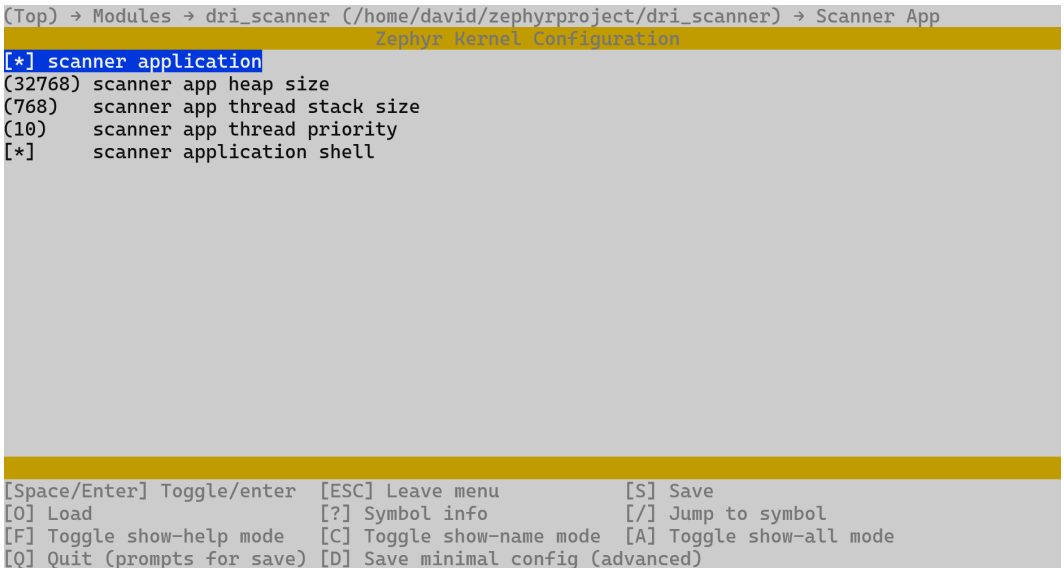
Výpis kódu 5.29 Konfigurace projektu *prj.conf*

Druhá možnost konfigurace vyžívá grafického nástroje menuconfig. Nástroj menuconfig lze spustit pomocí příkazu:

```
west build -t menuconfig
```

Výpis kódu 5.30 Spuštění nástroje menuconfig

Pokud bude aktivováno textové rozhraní aplikace, projekt využívající modul *dri_scanner* může obsahovat prázdnou funkci *main()*. Aplikace bude inicializována automaticky během startu systému.



Obrázek 5.2 Konfigurace modulu pomocí nástroje menuconfig

Kapitola 6

Testování a vyhodnocení

6.1 Jednotkové testy

Práce obsahuje základní sadu jednotkových testů, pomocí kterých je možné ověřit funkčnost jednotlivých částí aplikace. Jednotkovými testy je pokryta služba scanner, Wi-Fi a Bluetooth provider.

Projekt Zephyr poskytuje testovací framework ZTest, díky kterému je možné jednoduše psát jednotkové a integrační testy. Jednotlivé testy je možné spouštět jak na PC, kde je aplikace vyvíjena, tak i přímo na hardwarové platformě, pro kterou je aplikace psána.

Pro spouštění jednotkových testů slouží nástroj twister. Nástroj se postará o kompilaci jednotkových testů a případně je i nahraje na cílové zařízení. Pokud při spouštění testů není specifikována cílová platforma, twister vybere pouze takové testy, které lze pouštět na platformách podporovaných počítačem. Typicky jde o platformu *native_posix* či emulované prostředí *qemu_arm, qemu_x86*.

```
scripts/twister -T <test_directory>
```

Výpis kódu 6.1 Spuštění jednotkových testů

```
scripts/twister --device-testing --device-serial /dev/ttyUSB0 \
-p esp32c3_devkitm  -T <test_directory>
```

Výpis kódu 6.2 Spuštění jednotkových testů přímo na mikroprocesoru ESP32-C3

Nástroj twister bohužel po nahrání testů přímo na mikroprocesor ESP32-C3 nedokáže zařízení korektně resetovat, z toho důvodu končí test chybovou hláškou. Pokud se k zařízení připojíme pomocí sériové linky a zařízení restartujeme, uvidíme, že testy proběhly v pořádku.

6.2 Paměťová analýza

Nástroj west umožňuje vygenerovat analýzu využití paměti jednotlivých komponent aplikace. Lze analyzovat paměť RAM a ROM.

```
west build -t ram_report
```

Výpis kódu 6.3 Vygenerování analýzy paměti RAM

```
west build -t rom_report
```

Výpis kódu 6.4 Vygenerování analýzy paměti ROM

Před vygenerováním analýzy paměti RAM je vhodné znát aktuální konfiguraci modulu.

```
CONFIG_SCANNER_SERVICE=y

CONFIG_SCANNER_APP=y
CONFIG_SCANNER_APP_THREAD_PRIORITY=10
CONFIG_SCANNER_APP_HEAP_SIZE=32768
CONFIG_SCANNER_APP_SHELL=y

CONFIG_SCANNER_PROVIDER_BLUETOOTH=y
CONFIG_SCANNER_PROVIDER_ESP_WIFI=y
CONFIG_SCANNER_PROVIDER_MOCK=y
CONFIG_SCANNER_PROVIDER_MOCK_PRIORITY=20
CONFIG_SCANNER_PROVIDER_MOCK_DATA_COUNT=5
```

Výpis kódu 6.5 Konfigurace modulu během analýzy paměti RAM

Vytvořený modul vyžaduje k běhu 41880 KB paměti RAM, což je 9.7% z celkového nároku aplikace. Největší část využívá halda aplikační vrstvy, konkrétně 32768 KB (7.59%). V případě potřeby omezit využití paměti je možné pomocí Kconfig snížit její velikost. Dojde-li ke zmenšení haldy, sníží se maximální možný počet viditelných letounů. Do haldy s velikostí 32768 KB se vejdou informace o zhruba 60-ti dronech.

V paměti ROM zabírá modul 13088 KB. To je 1.92% z celkové velikosti aplikace. Detailní analýza vytvořeného modulu je přiložena v příloze práce.

6.3 Analýza příjmu

Uživatelské rozhraní umožňuje zobrazení statistik přijmu zpráv, příkaz *scanner statistics*. Statistiky ke každému zařízení zobrazují počet přijatých rámců, interval mezi jednotlivými rámci, počet přijatých unikátních zpráv (s rozdílným čítačem) a počet ztracených zpráv. Počet ztracených zpráv je spočten na základě stavu čítače naposledy přijaté zprávy a nově přijaté zprávy. Některá zařízení vysílají stejné zprávy vícekrát. Počet ztracených zpráv ukazuje zda-li nedošlo ke ztrátě informace, ale není podle toho možné zjistit počet ztracených rámců.

Další možnost, jak zjistit počet ztracených rámců, vyžaduje znalost intervalu odesílání zpráv. Statistiky zobrazují interval mezi posledními dvěma přijatými rámci, podle toho lze interval zjistit. Se znalostí intervalu odesílání je možné spočítat kolik zpráv má být za určitý časový úsek odesláno. Poté je možné porovnat počet reálně přijatých zpráv za stejný časový úsek s vypočtenou hodnotou.

Jako vysílač pro testování byla použita implementace pro ESP32 [1]. Vysílač odesílá zprávy pomocí Bluetooth zhruba každých 50 ms a pomocí Wi-Fi každých 800 ms. Celkový počet odeslaných zpráv bude 600, respektive 37. Příjem byl otestován při samostatném přijímání Bluetooth, samostatném přijímání Wi-Fi, přijímaní obou současně na jednom mikroprocesoru a naposledy přijímání obou současně při rozdělené architektuře Bluetooth Stack. Výsledky jsou následující:

Tabulka 6.1 Výsledky analýzy přijmu zpráv

	přijato		ztráty	
	Wi-Fi	**BT**	**Wi-Fi**	**BT**
pouze Wi-Fi	37	-	0 %	-
pouze Bluetooth	-	619	-	0 %
koexistence Wi-Fi a BT	18	351	51 %	41 %
rozdělená architektura	37	623	0 %	0 %

Výsledky testování Bluetooth příjmu bohužel nejsou přesné. Důvodem je, že interval mezi odesílanými zprávami není přesně 50 ms, ale docházelo ke kolísání kolem této hodnoty.

Dále bylo zjištěno, že při koexistenci technologií Wi-Fi a Bluetooth je vždy upřednostňován Bluetooth. Při nastavené velikosti okna Bluetooth skenování na 30 ms v rámci intervalu 60 ms (50 % času vyhrazeno pro Bluetooth) docházelo zhruba k polovičním ztrátám zpráv na obou technologiích. Pokud byla velikost okna zvětšena na 60 ms v rámci intervalu 60 ms (100 % času), nebylo možné přijímat téměř žádné Wi-Fi rámce.

Kapitola 7
Závěr

Cílem práce byla implementace aplikace umožňující příjem zpráv standardu ASD-STAN prEN 4709-002. V teoretické části práce byl shrnut standard ASD-STAN prEN 4709-002 a technologie Wi-Fi a Bluetooth, které se používají pro jejich přenos. Bylo provedeno srovnání jednotlivých mikroprocesorů výrobce Espressif Systems. Na základě srovnání byl zvolen mikroprocesor ESP32-C3. Pro vývoj aplikace byla použita vývojová deska ESP32-C3-DevKitM-1. Mikroprocesor podporuje bezdrátové technologie Bluetooth ve verzi 5, včetně rozšíření Extended Advertising a Bluetooth Long Range, a Wi-Fi verze 802.11/n.

K implementaci aplikace byl využit operační systém reálného času Zephyr. Aplikace je implementována jako modul do operačního systému, díky tomu je možné její použití v rámci většího celku. Architektura aplikace byla rozdělena do několika částí, aplikační vrstvy, služby scanner a providerů. Díky rozdělení byla zajištěna určitá úroveň hardwarové nezávislosti. Při změně platformy je třeba nahradit pouze ty části aplikace, které jsou hardwarově závislé. Hardwarově závislá komponenta aplikace je pouze Wi-Fi provider. Jednotlivé komponenty aplikace lze používat samostatně.

Mikroprocesor ESP32-C3 bohužel neobsahuje implementaci Wi-Fi Aware Engine, a tudíž neexistuje rozhraní pro snadný přístup ke zprávám odesílaným pomocí Wi-Fi Aware. Tento problém byl vyřešen přepnutím Wi-Fi mikroprocesoru do speciálního režimu, při kterém dochází k odchytávání všech přijatých rámců. Rámce, obsahující DRI zprávu, jsou dekódovány manuálně. Synchronizace zařízení v rámci Wi-Fi Aware skupiny nebyla potřeba řešit, protože k přijímání Wi-Fi rámců dochází nepřetržitě. Implementace přijímání zpráv pomocí technologie Wi-Fi využívá funkce specifické pro platformu ESP32. Pokud by někdy v budoucnu byla potřeba aplikaci provozovat na jiné platformě, bude potřeba implementovat nový provider umožnující přijímaní na nově zvolené platformě. K přijímaní Bluetooth zpráv je využit Bluetooh subsystém operačního systému Zephyr. Díky tomu je možné provozovat aplikaci přijímající Bluetooth zprávy na jakémkoliv zařízení s podporou Bluetooth v rámci operačního systému Zephyr.

ESP32-C3 obsahuje jednu sdílenou anténu mezi oběma technologiemi, díky tomu dochází ke ztrátám zpráv při současném přijímaní pomocí Wi-Fi a Bluetooth. Řešením tohoto problému bylo rozdělení příjímání mezi dva mikroprocesory ESP32-C3. Na jednom mikroprocesoru je provozovaná aplikace a příjem pomocí Wi-Fi. Příjem Bluetooth zpráv probíhá na druhém ESP32. Zařízení spolu komunikují pomocí transportního protokolu HCI UART.

V rámci práce byla vytvořena sada jednotkových testů. Testy pokrývají službu scanner, Wi-Fi a Bluetooth provider. Pro implementaci jednotkových testů byl zvolen framework ZTest. Testy je možné spouštět přímo na mikroprocesoru, případně na PC. Vyhodnocení ztrát během příjmu bylo provedeno jak při příjmu obou technologií zároveň na jednom mikroprocesoru, tak při rozdělení mezi dvě zařízení. Ukázalo se, že během přijímání pomocí jedné sdílené antény docházelo zhruba k 50% ztrátám na obou technologiích, tento problém se vyřešil při rozdělení přijmu mezi

dvě zařízení. Během testování bylo také zjištěno, že při koexistenci více bezdrátových technologií v rámci jednoho mikroprocesoru ESP32-C3 je vždy upřednostňována technologie Bluetooth. Nastavená délka okna Bluetooth skenování je vždy dodržena a Wi-Fi je možné přijímat pouze ve zbývajícím čase. Při nastavení nepřetržitého skenování Bluetooth nebylo možné přijímat téměř žádné Wi-Fi rámce.

Možným navázáním na tuto práci může být vytvoření vlastního zařízení s dvěma mikroprocesory ESP32-C3. Zařízení by mohlo komunikovat s PC, či mobilním telefonem, kde by se zobrazovaly viditelné drony na mapě. Případně je možné aplikaci rozšířit o podporu dalších standardů dálkové identifikace, například ASTM F3411.

Zkratky

ADS-B Automatic Dependent Surveillance–Broadcast.

AP Access point.

API Application programming interface.

ASCII American Standard Code for Information Interchange.

ASD-STAN Aerospace and Defense Industries Association of Europe - Standardization.

ASTM American Society for Testing and Materials.

AT Attention.

BLE Bluetooth Low Energy.

BT Bluetooth.

CID Company identifier.

DRI Přímá dálková identifikace.

FTM Fine Time Measurement.

GNSS Global navigation satellite system.

GPIO General-purpose input/output.

HAL Hardware abstraction layer.

HCI Host controller interface.

HW Hardware.

I2C Inter-integrated circuit.

ID Identifier.

IDE Integrated development environment.

IEEE Institute of Electrical and Electronics Engineers.

IPv4 Internet Protocol version 4.

IPv6 Internet Protocol version 6.

MAC Media access control address.

MP Mikroprocesor.

NAN Neighborhood awareness networking.

NRI Síťová dálková identifikace.

OGN Open Glider Network.

OOP Objektově orientované programování.

OS Operační systém.

OUI Organizationally unique identifier.

PC Personal computer.

RAM Random-access memory.

ROM Read-only memory.

RS-232 Recommended Standard 232.

RTOS Real-time operating system.

SDF Service discovery frame.

SoC System on chip.

SPI Serial peripheral interface.

SSID Service set identifier.

UART Universal asynchronous receiver-transmitter.

USB Universal Serial Bus.

UTM Unmanned aircraft systems traffic management.

Wi-Fi Wireless Fidelity.

Bibliografie

1. SXJACK. *uav_electronic_ids* [online]. [B.r.]. Dostupné také z: https://github.com/sxjack/uav_electronic_ids.
2. RICHARDS, William R.; O'BRIEN, Kathleen; MILLER, Dean C. New Air traffic Surveillance Technology. *AERO* [online]. 2010, roč. 2010, č. Q2, s. 7–13. Dostupné také z: https://www.boeing.com/commercial/aeromagazine/articles/qtr_02_10/pdfs/AERO_Q2-10.pdf.
3. ŘÍZENÍ LETOVÉHO PROVOZU ČESKÉ REPUBLIKY, S. P. *U-space - Létejte zodpovědně* [online]. Dostupné také z: https://www.letejtezodpovedne.cz/legislativa/co_nas_ceka?clid=268.
4. ASD-STAN. *DIN EN 4709-002:2021-02* [online]. 2020. Tech. zpr. AeroSpace and Defence Industries Association of Europe. Dostupné také z: https://asd-stan.org/downloads/din-en-4709-0022021-02/.
5. ASD-STAN. *Introcuction To The European UAS Digital Remote ID Technical Standard* [online]. 2021. Dostupné také z: https://asd-stan.org/wp-content/uploads/ASD-STAN_DRI_Introduction_to_the_European_digital_RID_UAS_Standard.pdf.
6. OPENDRONEID. *Open Drone ID Core C Library* [online]. 2021. Ver. 1.0. Dostupné také z: https://github.com/opendroneid/opendroneid-core-c.
7. DRONETAG S.R.O. *Dronetag Mini* [online]. 2022. Dostupné také z: https://dronetag.cz/products/mini/.
8. IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*. 2021, s. 1–4379. Dostupné z DOI: 10.1109/IEEESTD.2021.9363693.
9. WI-FI ALLIANCE. *Wi-Fi CERTIFIED Wi-Fi Aware™ Technology Overview* [online]. 2020. Dostupné také z: https://www.wi-fi.org/downloads-registered-guest/Wi-Fi_CERTIFIED_Wi-Fi_Aware_Technology_Overview_202012.pdf/35380.
10. WI-FI ALLIANCE. *Wi-Fi Aware™ Specification* [online]. 2020-10-23. Ver. 3.2. Tech. zpr. Dostupné také z: https://www.wi-fi.org/downloads-registered-guest/Wi-Fi_Aware_Specification_v3.2.pdf/29731.
11. BLUETOOTH SIG, INC. *Bluetooth Core Specification* [online]. 2021-07-13. Ver. 5.3. Tech. zpr. Dostupné také z: https://www.bluetooth.com/specifications/specs/core-specification-5-3/.

12. WOOLLEY, Martin. *Bluetooth® Core Specification Version 5.0 Feature Enhancements* [online]. 2021-09-09. Ver. 1.1.0. Tech. zpr. Bluetooth SIG, Inc. Dostupné také z: https://www.bluetooth.com/wp-content/uploads/2019/03/Bluetooth_5-FINAL.pdf.
13. SUN, Junzi. *The 1090 Megahertz Riddle: A Guide to Decoding Mode S and ADS-B Signals*. 2. vyd. TU Delft OPEN Publishing, 2021. ISBN 978-94-6366-402-8. Dostupné z DOI: 10.34641/mg.11.
14. *Open Glider Network Project* [online]. 2022. Dostupné také z: http://wiki.glidernet.org/.
15. OPENDRONEID. *OpenDroneID Android receiver application* [online]. 2021. Ver. 3.0.0. Dostupné také z: https://github.com/opendroneid/receiver-android.
16. ARDUINO. *Arduino IDE* [online]. 2016. Ver. 1.8.19. Dostupné také z: https://www.arduino.cc/en/software.
17. OPENDRONEID. *Open Drone ID transmitter example for Linux* [online]. 2021. Ver. 1.0. Dostupné také z: https://github.com/opendroneid/transmitter-linux.
18. THE LINUX FOUNDATION. *Zephyr Project Documentation* [online]. 2022. Ver. 3.0.0. Dostupné také z: https://docs.zephyrproject.org/3.0.0/.
19. ESPRESSIF SYSTEMS. *ESP-IDF Programming Guide* [online]. 2022. Dostupné také z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/index.html.
20. ESPRESSIF SYSTEMS. *ESP32-C3 Series Datasheet* [online]. 2022. Ver. 1.2. Tech. zpr. Dostupné také z: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf.
21. LI, Qing; YAO, Caroline. *Real-Time Concepts for Embedded Systems*. 1. vyd. Taylor & Francis Ltd, 2003. ISBN 1578201241.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `mem_analysis` ..... paměťová analýza modulu
- `screens` ..... obrazovky uživatelského rozhraní
- `src`
  - `dri_scanner` ..... zdrojový kód vytvořeného modulu
  - `esp_bt_controller` ..... nakonfigurovaný projekt pro BT controller
  - `scanner_application` ..... projekt využívající modul *dri_scanner*
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_David_Horak_2022.pdf` ..... text práce ve formátu PDF