
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**
**DEPARTMENT OF COMPUTER SYSTEMS**


# BEZDRÁTOVÝ LOKALIZAČNÍ MODUL S NÍZKO-PŘÍKONOVÝM FIRMWARE NA BÁZI RTOS

**WIRELESS LOCALIZATION MODULE WITH LOW-POWER FIRMWARE BASED ON RTOS**

**DIPLOMOVÁ PRÁCE**
**MASTER'S THESIS**


**AUTOR PRÁCE** **Bc. RADIM LIPKA**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Ing. VÁCLAV ŠIMEK**
**SUPERVISOR**



**BRNO 2020**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

# Zadání diplomové práce


Student: **Lipka Radim, Bc.**
Program: Informační technologie Obor: Počítačové a vestavěné systémy
Název: **Bezdrátový lokalizační modul s nízko-příkonovým firmware na bázi RTOS**
**Wireless Localization Module with Low-Power Firmware Based on RTOS**
Kategorie: Vestavěné systémy

Zadání:

1. Prostudujte a porovnejte vlastnosti aktuálně dostupných RTOS pro vestavěná zařízení. Zaměřte se především na techniky pro dosažení velmi nízké spotřeby elektrické energie.
2. Seznamte se s teoretickými principy lokalizace objektů v bezdrátových komunikačních sítích. Detailně se zabývejte především technologií UWB.
3. Navrhněte a na obvodové úrovni zrealizujte hardware tagu, který bude obsahovat rádiový modul UWB, mikrokontrolér ARM M4, akcelerometr a E-INK displej.
4. Implementujte ovladače pro lokalizační modul pod vybraným RTOS. Dále vytvořte firmware umožňující vysílání lokalizační zprávy v různých módech a zobrazení provozních informací na E-INK displeji.
5. S využitím dostupných vývojových prostředků pro zvolený typ mikrokontroléru měřte a vyhodnoťte energetickou náročnost částí implementovaného firmwaru, které se týkají lokalizace.
6. Dále analyzujte stabilitu firmwaru, a to zejména s ohledem na okrajové jevy jako například "race-condition" v době přechodu mikrokontroléru do hlubokého spánku.
7. Zhodnoťte dosažené výsledky, diskutujte možnosti dalšího rozvoje a případná vylepšení.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Šimek Václav, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 20. května 2020
Datum schválení: 25. října 2019

# Abstrakt

Práce se zabývá návrhem a implementací bezdrátového lokalizačního modulu, který využívá technologii UWB v kombinaci s nízko-příkonovým firmware na bázi vestavěného RTOS systému. Lokalizace modulu probíhá za pomocí algoritmu TDoA. Výsledný HW modul je realizován jako vývojová dvouvrstvá DPS, založená na MCU nRF52832 (ARM, Cortex M4) a UWB modulu DecaWave DW1000. Výsledný firmware je implementován pod FreeRTOS systémem s důrazem na nízkou spotřebu. K návrhu HW je použit Eagle CAD a k implementaci firmware programovací jazyky C a Assembler.



# Abstract

This thesis focuses on the design and implementation of the wireless localization module, using UWB technology with emphases on low-power firmwre based on RTOS. Wireless localization is based on TDoA algorithm. The resulting HW module is designed as a four layer PCB, based on MCU crf52832 (ARM Cortex M4) and UWB module DevaWave DW1000. Firmware is implemented using FreeRTOS with emphasis on low power consumption. For hardware implementation, Eagle CAD was used. Firmware is implemented in C and Assembly programming languages.



# Klíčová slova

UWB, FreeRTOS, bezdrátová lokalizace, nízký příkon, DPS, nRF52832, Cortex-M4, TDOA, DWM1001, DW1000

# Keywords

UWB, FreeRTOS, wireless localization, low-power, PCB, nRF52832, Cortex-M4, TDOA, DWM1001, DW1000


# Citace

LIPKA, Radim. *Bezdrátový lokalizační modul s nízko-příkonovým firmware na bázi RTOS.* Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Václav Šimek

# Bezdrátový lokalizační modul s nízko-příkonovým firmware na bázi RTOS

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Václava Šimka. Další informace mi poskytl pan Ing. Lubomír Mráz. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Radim Lipka
10. června 2020


## Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Václavu Šimkovi za užitečné rady, konzultace, pomoc s HW realizací tagu a přátelskou spolupráci. Také děkuji Ing. Lubomíru Mrázovi, CTO společnosti Sewio Networks s.r.o., za zajímavou příležitost k rozvoji.

# Obsah

# Kapitola 1

# Úvod

Bezdrátová lokalizace slouží k určení polohy sledovaného objektu v prostoru. Významně zjednodušuje a zefektivňuje procesy v mnoha aspektech života, především v navigování a sledování. V otevřeném prostoru je nejčastěji používanou metodou GPS (korektně pak GNSS), její nevýhoda však spočívá v nedostatečné síle signálu. Ten nedokáže proniknout dovnitř budov, ve kterých je lokalizace taktéž potřebná.

Určování polohy uvnitř budov je využíváno v nemocnicích, průmyslových halách, letištích, nákupních střediscích a dalších prostorech. Slouží například ke sledování pacientů, drahého vybavení, navigování lidí s poruchami vidění, k optimalizaci výroby, záchranným operacím a dalším úkonům. Využití lokalizace uvnitř budov je neustále rostoucí oblastí vývoje. Důraz je kladen na vysokou přesnost, nízkou cenu, dobrou dostupnost, vysoké pokrytí, rozšiřitelnost a soukromí.

Tato práce, jejíž zadání bylo připraveno společností Sewio Networks s.r.o., se zabývá vývojem hardwaru i firmwaru bezdrátového vývojového lokátoru (tagu), který může být lokalizován pomocí rádiové technologie UWB. Významným aspektem je v případě zařízení důraz kladený na optimalizaci obvodových i programových zdrojů za účelem dosažení nízké spotřeby.

Kapitola 2 je věnována zasvěcení do technologií a algoritmů, které umožňují lokalizaci uvnitř budov. Mimo jiné popisuje vlastnosti, principy a výběr operačního systému reálného času, který je posléze použit jako řídící program pro výsledný lokátor. Následuje kapitola 3, která se věnuje specifikaci, návrhu a výrobě hardwaru tagu. Kapitola 4 popisuje specifikaci a návrh řídícího programu (firmware) za použití operačního systému reálného času. Následuje popis implementace, kde jsou vysvětleny detaily výsledného programu. V kapitole 5 je vývojová deska otestována a je vyhodnocena její spotřeba. Závěrečná kapitola 6 se věnuje zhodnocení dosažených výsledků a navržení možných vylepšení.

# Kapitola 2

# Použité technologie

Tato kapitola popisuje základní znalosti, které jsou potřebné k hlubšímu pochopení problematiky. Zabývá se metodami bezdrátové lokalizace, jejími algoritmy a podrobně popisuje bezdrátovou technologii UWB. Dále pak popisem RTOS operačních systémů a jejich principů, které jsou použity pro výslednou implementaci lokátoru.

## 2.1 Bezdrátová lokalizace

Jedná se o metodu, která za pomoci dané bezdrátové technologie umožňuje získat pozici sledovaného objektu. Obecně je možné technologie lokalizace rozdělit do kategorií podle prostředí, ve kterých lokalizaci provádíme [1].

První z nich je lokalizace uvnitř budov a v městském prostředí (indoor). V tomto prostředí technologie umožňuje navigovat lidi s poruchami vidění, sledovat drahé vybavení a zabránit tak krádežím, pomáhat sportovcům s tréninkem, sledovat pacienty v nemocnicích, poskytovat podporu záchranářům, či optimalizovat procesy ve výrobních halách nebo skladech.

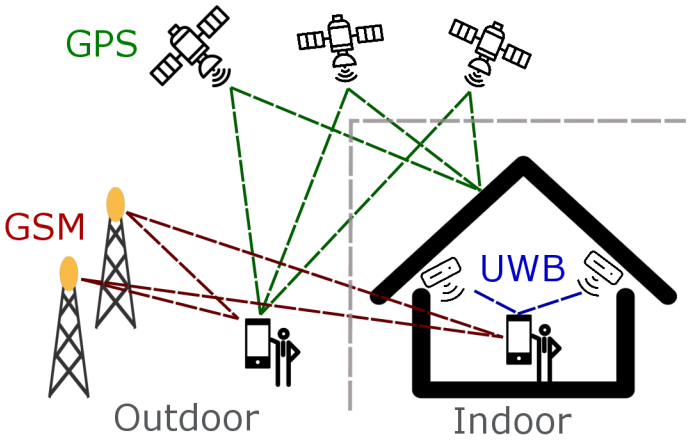
Druhou kategorií jsou venkovní prostředí (outdoor), kde je technologie využita primárně k navigování. Běžní lidé se s ní setkají např. při jízdě na dovolenou nebo navigování k obchodu. V profesionální sféře technologie využívají např. hasiči, policie, záchranná služba, a armáda.

V této podkapitole následuje popis technologií a metod pro lokalizaci s důrazem na technologii UWB. Na obrázku 2.1 lze vidět zjednodušený rozdíl mezi vnitřní (indoor) a vnější (outdoor) lokací.

### 2.1.1 Lokalizace ve venkovním prostředí

Nejpoužívanější metodou pro venkovní (outdoor) lokalizaci je GPS. Tento pojem ovšem není přesný. Přešel do používání v obecné mluvě pro označení jakéhokoliv systému pro zjišťování polohy. GPS je nicméně systém provozovaný pouze ministerstvem obrany USA. Správné označení pro technologie zaštiťující rádiové zjišťování polohy pomocí družic s celosvětovým pokrytím je globální družicový polohový systém (GNSS) [2]. Existuje celá řada GNSS systémů, které fungují na stejném principu jako GPS. Ruská armáda používá GLONASS, v Evropské unii je vyvíjen Galileo a Čínská lidová republika pracuje na vývoji BeiDou.

Tyto systémy jsou schopny zjistit polohu v přesnosti na desítky metrů až jednotky centimetrů. Jejich hlavní nevýhodou je ovšem síla signálu, která rapidně klesá vlivy pro-




Obrázek 2.1: Lokalizace ve vnějším (outdoor) a vnitřním (indoor) prostředí

středí, přes které prochází. Není tudíž vhodná a použitelná pro navigování uvnitř většiny uzavřených prostorů. Podle [3] je přesnost GPS ve většině uzavřených objektů 5-50 m [2].

### 2.1.2 Lokalizace ve vnitřním prostředí

Pro účely zjišťování polohy objektů uvnitř budov je potřeba prozkoumat metody, označované jako Indoor Positioning Systems (IPS). Značný rozdíl oproti metodám lokalizace ve vnějším prostředím pak spočívá v jejich vyšší komplexnosti, neboť uvnitř budov se nachází spousta objektů (nábytek, vybavení, lidé), které signál ovlivňují.

Odraz (reflection), vede k problémům se zpožděním (delay) a k vícecestnému šíření signálu[1]. Další skutečností je, že zdroj signálu není vždy v přímé viditelnosti s příjemcem (non line of sight), což způsobuje další problémy a zvyšuje složitost detekce. Existence objektů dále zapříčiňuje rozptyl (scattering) signálů a jejich pohlcování a tedy zeslabení (attenuation). Průběh signálu může být dále zarušen velkým množstvím dalších současně použitých technologií (interference). Další nevýhodou je možnost častého přesouvání objektů, nebo samotných referenčních bodů. Oproti metodám lokalizace ve venkovních prostorech je zde vyžadována daleko vyšší přesnost, dobrá dostupnost služby, vysoké pokrytí prostoru, snadná rozšiřitelnost, soukromí a samozřejmý je také tlak na nízkou cenu [1].

V následujících podkategoriích jsou technologie rozděleny podle média, které je použito pro přenos informace k určení polohy. Rádiové (RFID, UWB, Wi-Fi, Bluetooth, mobilní sítě), akustické (ultrazvuk), světelné (infračervené záření) a senzorové (dead reckoning) [1]. Následuje tabulka 2.1, která vybrané technologie srovnává.

#### Radio Frequency Identification (RFID)

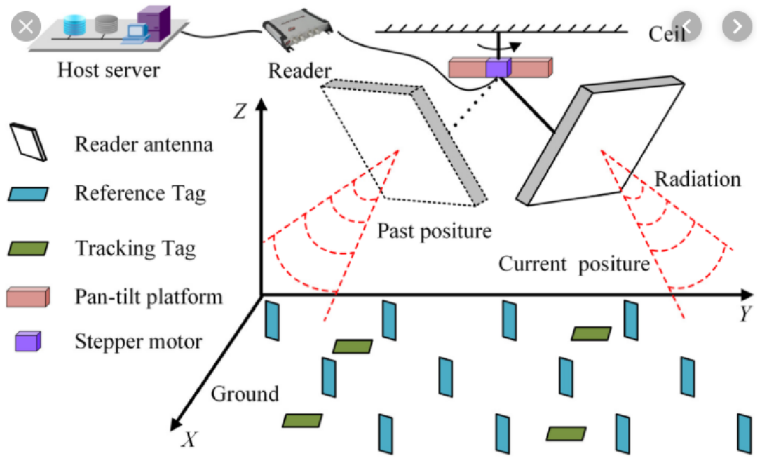
Tato technologie využívá bezdrátového rádiového přenosu k identifikaci objektu. Pracuje na frekvencích 125 kHz, 134 kHz, 13,56 MHz, 868 MHz (Evropa) a 915 MHz (Amerika). Tyto systémy využívají dvou komponent.

[1] Vícecestné šíření (Multipath propagation) je jev, který vzniká odrážením (reflection) a rozptylem (scattering) rádiového signálu od okolních objektů, což zapříčiňuje přijetí několika stejných signálů s různou fází, nebo rušení (interference) signálu.

Prvními jsou tagy, které jsou připevněny na všechny objekty, které mají být sledovány. Skládají se z mikročipu, který umožňuje uložit okolo 2 kB dat, a rádiové antény. Existují dva typy tagů, aktivní a pasivní, dle toho, zda tag obsahuje vlastní zdroj napájení, nebo musí být nabit čtečkou.

Druhým objektem jsou čtečky dodávající energii pasivním tagům, které typicky vyšlou informaci uloženou ve své paměti, nebo jinak ovlivní modulaci výstupního signálu. Čtečky se skládají z antény, vysílače, přijímače, zdroje, procesoru a rozhraní k připojení do sítě.

Tato metoda je spíše využívána ke zjišťování, zda se objekt nachází či nenachází poblíž čtečky, ale může být využita i informace o RSS (síla signálu, viz podkapitola 2.1.3) pro určení vzdálenosti. Výhodou technologie je, že nepotřebuje přímou viditelnost mezi vysílačem a přijímačem. Nevýhodami jsou pak nízký dosah (v případě pasivního tagu 2 metry, u aktivního pak 10 metrů) a vysoká spotřeba [3] [1]. Na obrázku 2.2 lze vidět možnou architekturu RFID lokalizačního systému. Bílými objekty jsou antény, napojené na čtečky. Modrými objekty jsou sledované tagy. Naměřená data jsou odesílána a dále zpracovávána na server.



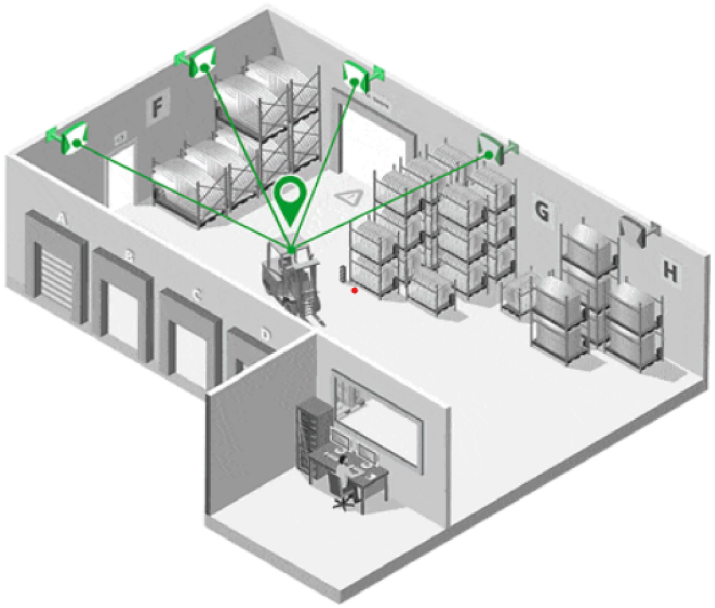

Obrázek 2.2: RFID architektura, převzato z [4]

### Ultra Wideband (UWB)

Touto technologií se podrobněji zabývá podkapitola 2.1.4. Ve zkratce se jedná o rádiovou technologii, která využívá krátkých pulzů ($< 1$ ns) ke komunikaci skrz velmi široké pásmo (větší, než 500 MHz), v pásmu 3,1-10,6 MHz. Díky tomu, že je pulz rozprostřen skrz celé pásmo, se podobá šumu, který téměř neruší ostatní technologie a je odolný vůči některým negativním vlivům rádiového přenosu.

Výhodou je vysoká přesnost, schopnost průniku skrz objekty a odolnost vůči vícecestnému šíření. Mezi nevýhody potom patří vysoká cena a snadná zarušitelnost kovovými objekty nebo objekty obsahující vodu (například lidské tělo).

K určení vzdálenosti může být použita metoda TOA (viz podkapitola 2.1.3). Zařízení, které je lokalizováno je dále nazýváno tagem. Zařízení, která lokalizaci provádějí jsou pak nazývaná kotvami [3]. Na obrázku 2.3 lze vidět kotvy na stěnách místnosti, lokalizující tag umístěný na vysokozdvižném vozíku.




Obrázek 2.3: UWB sledování, ukázka z praxe, převzato z [5]

### Infrared (IR)

Neviditelné infračervené záření je elektromagnetické záření s velikostí vlnové délky od 700 nm-1 mm (430 THz-300 GHz). Výhodou oproti použití viditelného světla v lokalizaci je právě jeho neviditelnost, která neruší okolí. Základem je aktivní tag, který je nošen uživateli a vysílá periodický signál (globální unikátní ID), který je zachycen fixními přijímači. Existují dvě implementace této technologie.

První z nich je přímé IR (DirectIR, IrDA). To vyžaduje přímou viditelnost mezi vysílačem a přijímačem. Na krátké vzdálenosti umožňuje komunikovat rychlostí až 16 Mb/s.

Druhou je diffuse IR, které využívá vyšší sílu signálu s větším dosahem 9-12 m. Využívá LED, které umožňují vysílání signálů do více směrů, nevyžaduje tedy přímou viditelnost s přijímačem, neboť může zároveň vysílat k několika z nich.

Nejčastěji používanou metodou lokalizace bývá AoA (viz podkapitola 2.1.3). Nevýhodou je omezení použití na prostor jediné místnosti, neboť signál nemůže proniknout skrz zdi. Dále je velmi lehce blokován překážkami a může být rušen slunečním zářením [1] [6].

### Ultrazvuk (Ultrasonic)

Ultrazvukové akustické vlnění je mechanické vlnění s frekvencí na pomezí slyšitelného zvuku okolo 20 KHz. Díky tomu, že se jedná o zvuk, nekoliduje s elektromagnetickým vlněním.

Systém Active Bat [6], funguje na základě odeslání rádiových pulzů tagům. Tagy ihned po zachycení pulzu produkují ultrazvukový signál, který je zachycen rádiovými vysílači na stropech a zdech. Výsledná pozice může být vypočítána multilaterací[2] krátkých pulzů vysílaných od zdroje k přijímačům.

[2]Multilaterace je metoda zjišťování polohy sledovaného objektu za pomoci N stanic se známými pozicemi.

Výhodou je, že nevyžaduje přímou viditelnost mezi vysílačem a přijímačem, nevýhodou pak falešné signály způsobené odrazy a rušení zvukem vysoké frekvence [1].

### Wireless Local Area network (WLAN)

Tato rádiová technologie je nejpoužívanější bezdrátová rádiová technologie určená pro komunikaci v lokálních bezdrátových sítích a přístup k internetu. Je tomu tak především pro její velký dosah a umožnění vysoké komunikační rychlosti. Je podporována velkým množstvím spotřebních zařízení. Díky těmto faktům je její dostupnost obrovská, převážně pak v uzavřených budovách a městských prostorech.

WPS (Wi-Fi-based positioning system) [7] je metoda, která využívá bezdrátové přístupové body (access points) k určení pozice lokalizovaného zařízení. K určení vzdálenosti se používají metody RSSI (viz podkapitola 2.1.3), ToF, nebo AoA. K určení pozice pak kombinace různých metod (hybridní přístup), multilaterace, nebo fingerprinting (viz podkapitola 2.1.3). Přesnost záleží především na počtu nejbližších přístupových bodů a na prostředí.

Ačkoli je původním účelem Wi-Fi sítí především zprostředkování komunikace, ani jejich užitečnost při lokalizaci není zanedbatelná. Lze totiž dosáhnout přesnosti až 23 cm [8]. Výhoda tohoto přístupu je, že není potřeba budovat novou infrastrukturu, neboť ji s dobrým škálováním můžeme jednoduše rozšířit [1].

### Bluetooth

Bluetooth (IEEE 802.15.1), podobně jako Wi-Fi, pracuje na frekvenci 2,4 GHz. Umožňuje lokalizaci pomocí ToF i AoA, momentálně je ovšem nejvíce používáno RSSI. Původně nebylo Bluetooth a BLE (Bluetooth Low Energy) k účelu lokalizace navržené.

Nevýhodou je nízký dosah a nutnost dostatečného počtu vysílačů k přesnému určení pozice. Výhodou oproti Wi-Fi je jeho menší energetická náročnost [8][1].

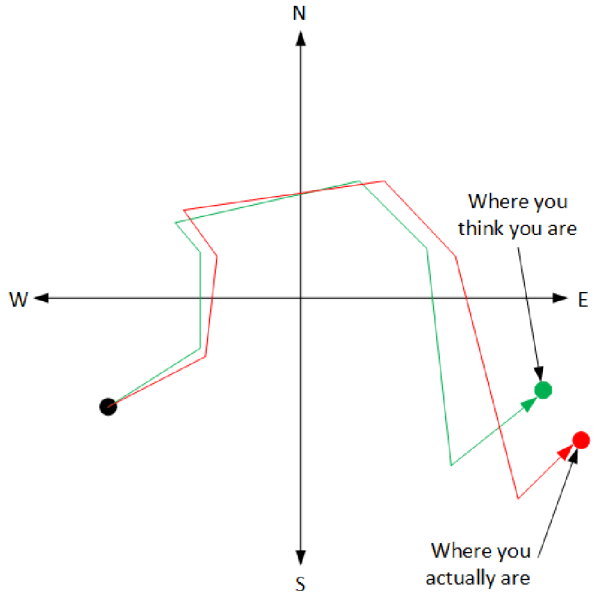
### Mobilní sítě (Cellular based)

GSM (Global System for Mobile Communications) mobilní sítě jsou dostupné na většině území. Slouží k výměně datových zpráv a ke komunikačním účelům a dokáží zpřesnit zde uvedené metody.

Výhodou je, že GSM funguje na licencovaných pásmech, která jsou hůře zarušitelná jinými technologiemi. K lokaci se používá RSSI, které může být kombinováno s algoritmy laterace, nebo fingerprintingu. Dá se použít i k lokalizaci ve vnitřních prostorech, přesnost měření je pak závislá na počtu a hustotě okolních základových stanic [9] [1].

### Dead Reckoning

Za použití inerciálních jednotek (akcelerometr, gyroskop, magnetometr) a barometru dokáže objekt určovat aktuální pozici za použití pozice předchozí. Při započetí navigování musíme znát přesnou pozici objektu. Nevýhodou je neustálá kumulace chyb i při malých odchylkách (viz obrázek 2.4, kde zelená čára značí trajektorii vypočítanou senzory, červená čára pak skutečnou trajektorii). Výhodou je, že nevyžaduje žádné další senzory a přepracování infrastruktury [1].




Obrázek 2.4: Dead reckoning, kumulování chyby a realita, převzato z [10]

| **Technologie** | **Systém** | **Dosah** | **Přesnost** |
|---|---|---|---|
| IR | Active Badge | 5 m | 7 cm |
| Ultrazvuk | Active Bat | 30 m | 9 cm |
| RFID | LANDMARCE | 35 m | 2-5 m |
| UWB | Ubisense | 20 m | 10 cm |
| WLAN | Horus | 50 m | 2-3 m |

Tabulka 2.1: Srovnání technologií (2011), převzato z [6]

### 2.1.3 Metody určení lokace uvnitř budov

Zjišťování polohy je typicky realizováno ve dvou krocích. Nejprve proběhne měření vlastností signálu (doba, úhel, síla, fáze) mezi lokalizovaným zařízením a systémem a následuje výpočet pozice zařízení. Je třeba pamatovat na to, že signál vždy putuje za pomoci daného média (rádiové vlny, světlo, zvuk), které má různé vlastnosti. Měřené vlastnosti signálu mohou určovat i následně zvolený algoritmus použitý k výpočtu výsledné pozice [6] [8] [1].

#### Time of arrival (ToA, ToF)

Měřenou vlastností je čas přenosu signálu mezi vysílačem a přijímačem, který je převeden na vzdálenost [11]. Tento čas je získán jako rozdíl času zachycení signálu příjemcem $T_p$ a času odeslání signálu odesílatelem $T_o$. Vzdálenost je vypočítána za znalosti rychlosti šíření signálu daným médiem $V_m$ jako (2.1):

$$S = (T_o - T_p)/V_m \tag{2.1}$$

Výsledná lokace zařízení je spočítána pomocí trilaterace/multilaterace (viz obrázek 2.5 – a), dle měření vzdálenosti od více buněk s přesně danými souřadnicemi. K určení

pozice ve 2D prostoru jsou zapotřebí tři přijímače, v 3D prostoru pak čtyři. Nevýhodou tohoto přístupu je přesná synchronizace mezi vysílači a přijímači.

Tato metoda je využívána například v GPS, kdy měříme dobu letu signálu od družic k našemu GPS zařízení.

*Two way Ranging (TwR)* je metoda, která také může vést k určení ToF [12]. Předpokladem je obousměrná komunikace vysílače a přijímače. Tag si vymění s kotvou dvě zprávy. Tag si poznačí čas odeslání $T_1$ první zprávy a kotva čas jeho přijetí $T_2$. Následně kotva vyšle novou zprávu a poznačí si čas odeslání $T_3$, přičemž kotva po přijetí zaznačí tento čas jako $T_4$. Výsledný čas přenosu zprávy mezi tagem a kotvou je poté spočítán z rovnice 2.2).

$$T = ((T_4 - T_1) - (T_3 - T_2))/2 \tag{2.2}$$

### Time differences of arrival (TDoA)

Měřenou vlastností je čas přijetí na více přijímačích, které znají svou pozici a jejich hodiny jsou synchronní. Tato metoda funguje pomocí odeslání zprávy na tagu a přijetím té samé zprávy na více kotvách. Při výpočtu pozice není potřeba znát čas odeslání impulzu, ale hodiny příjemců musí být synchronizované. Následně dochází k výpočtu rozdílů přijetí zprávy mezi všemi kotvami, jejichž pozice je známa. Z těchto rozdílů je spočítán poměr vzdálenosti mezi danými kotvami a tagem. Z nich je vypočítána pozice tagu, vypočítáním průsečíku hyperbol (viz obrázek 2.5 – b). K určení pozice ve 2D prostoru jsou zapotřebí tři přijímače, v 3D prostoru pak čtyři.

### Angle of arrival (AoA)

Měřenou vlastností je úhel příjmu signálu odesílaného odesílatelem na poli antén přijímače. Výsledná poloha zařízení se vypočítá jako průsečík přímek od všech přijímačů ve směru jejich přijatých signálů (viz obrázek 2.5 – c). Pozice přijímačů musí být předem známá. Vypočítávání výsledné polohy se nazývá triangulace.

Chyba této metody roste se vzdáleností objektu, kdy je kladen velký důraz na rozlišení antén. Ty musí být velmi přesné a jsou tudíž velmi drahé. Zpřesnění výpočtu je možné dosáhnout například přidáním více detektorů. Další nevýhodou je rušení způsobené vícecestným šířením a nutnost přímé viditelnosti mezi odesílatelem a přijímačem. Z toho důvodu se tato metoda používá převážně v otevřeném prostoru. K detekci ve 2D prostoru jsou zapotřebí dva přijímače, v 3D prostoru pak tři.

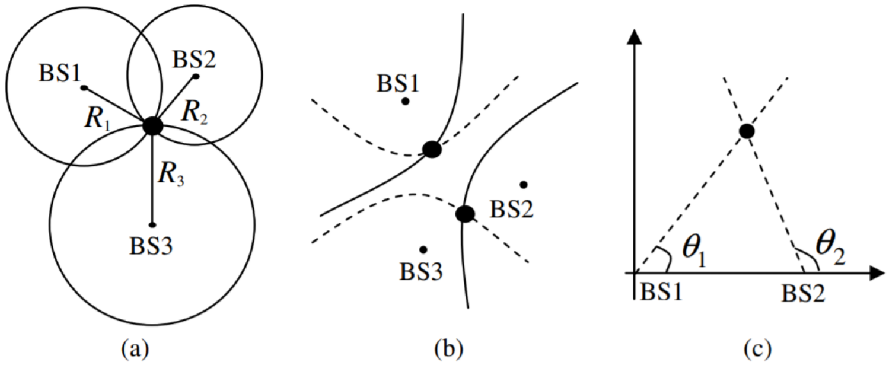
### Received signal strength (RSS)

Měřenou vlastností je síla přijatého signálu měřená na přijímači, která se s rostoucí vzdáleností $d$ snižuje, jak je patrno z rovnice 2.3. Ztrátový koeficient je poté $n$. Pro volný prostor je $n = 2$.

$$RSS = 1/d^n \tag{2.3}$$

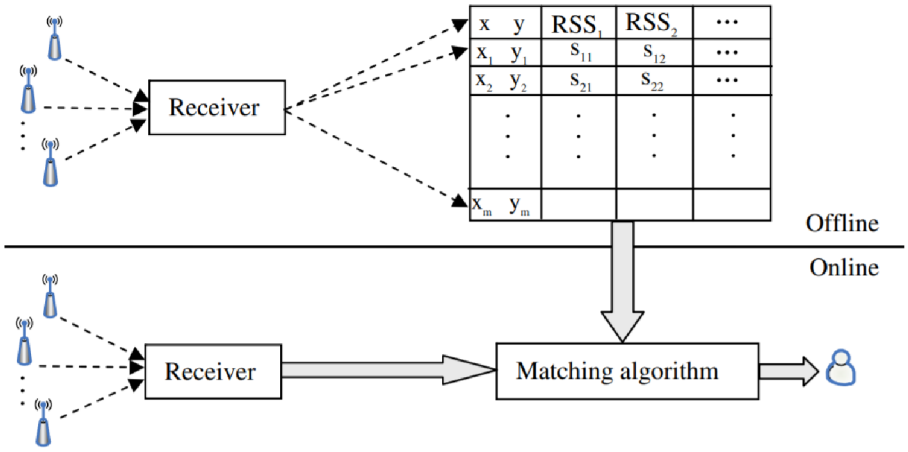
Received signal strength indicator (RSSI) je poté síla přijatého signálu v rádiové komunikaci. Existují dva algoritmy, které dokáží odhadnout pozici, nebo vzdálenost zařízení z RSSI [11].

*Propagation model algorithm (PMA)* pracuje nad převodem mezi sílou signálu a vzdáleností. Na základě síly signálu je možné pomocí modelu získat adekvátní vzdálenost. Metoda dobře funguje v otevřených prostorech, kde je ztráta síly signálu adekvátní ke vzdálenosti. V uzavřených prostorech, kde jsou překážky a dochází k útlumu signálu, již toto neplatí.




Obrázek 2.5: Lokalizační techniky: (a) TOA, (b) TDOA, (c) AOA, převzato z [6]

*Fingerprinting algoritmus* má dvě fáze. První je offline trénování, ve kterém dochází k mapování údajů ze senzorů vůči daným pozicím. Pozice a hodnoty se ručně zaznamenávají do tabulky (vrchní polovina obrázku 2.6), kde v jedné ose jsou aktuální pozice a v ose druhé síly signálů od fixních vysílačů. Druhou fází je online fáze, kde dochází k naměření aktuálních hodnot sil signálů ke všem vysílačům a k vyhledání nejbližší hodnoty v tabulce (dolní polovina obrázku 2.6), která určí výslednou pozici. Nevýhodou tohoto přístupu je nutnost provést aktualizaci tabulky v případě větší změny podmínek (přesun vysílače, nábytku, stínění, . . . ).

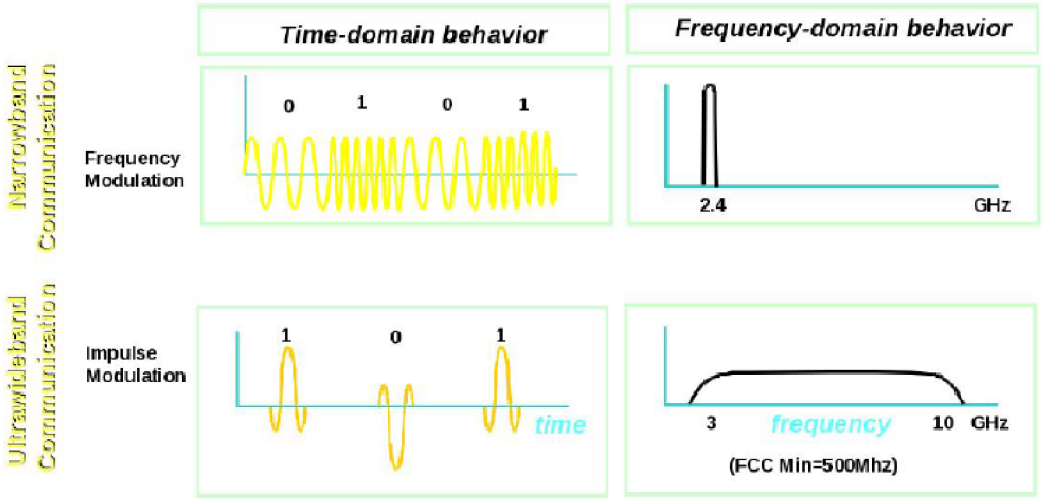



Obrázek 2.6: Ukázka metody fingerprintingu, převzato z [6]

### 2.1.4 Ultra Wide-band (UWB)

Historie UWB sahá do 40. let 20. století, kdy byla poprvé používána v radarové technice. DARPA (Defense Advanced Research Project Agency) jej však poprvé definovala teprve v roce 1990. Později se stala i komerčně dostupnou [13] [14].

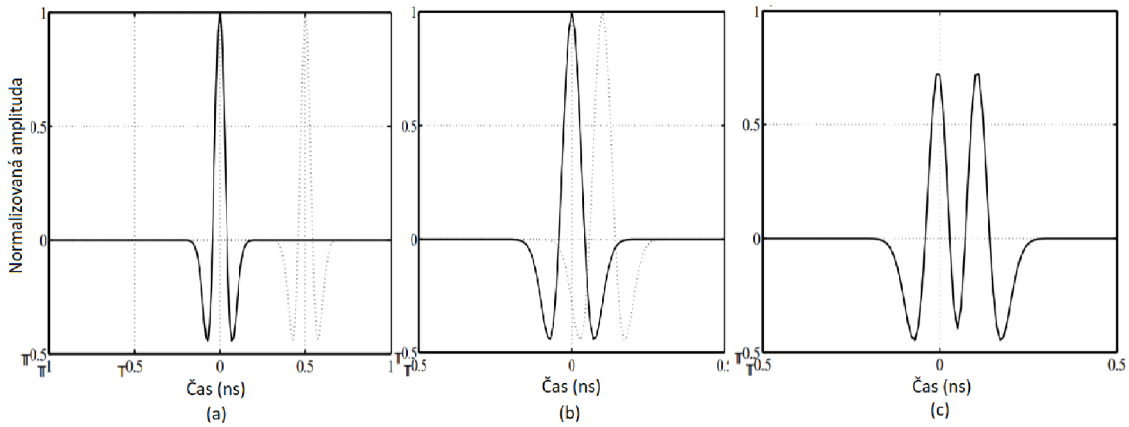
Oproti jiným úzkopásmovým technologiím je UWB komunikace charakteristická šířkou pásma větší než 500 MHz, v rozmezí 3,1-10,6 GHz. Omezená je pak maximálním výkonem, který činí -41.3 dBm/MHz. Z časové domény úzkopásmových technologií, kdy kmitajícím signálem dostáváme impulz v doméně frekvenční, UWB namísto modulování kontinuální nosné moduluje v časové doméně impulzy, které jsou ve frekvenční doméně charakterizovány jako oblasti frekvencí (viz obrázek 2.7, kde horní řádek představuje úzko-frekvenční technologie, dolní pak ultra-wideband. Levá polovina představuje časovou doménu, pravá polovina frekvenční).




Obrázek 2.7: Rozdíl mezi narrowband a ultra-wideband, převzato z [15]

Tyto impulzy mohou mít několik tvarů, většinou se jedná o variace derivací gaussova impulzu. Šířky impulzů se pohybují v rozmezí 0,2-1,5 mm. Díky velmi široké využité šířce pásma a malému vyzařovacímu výkonu jsou UWB signály detekovány úzkopásmovými technologiemi jako šum, který je téměř nemůže zarušit a komunikace je náročná na odposlech. Dále je uváděno, že UWB komunikace nemůže zarušit další technologie, je levná, rychlá a má nízkou spotřebu.

UWB systémy jsou považovány za imunní vůči vícecestnému šíření, které může způsobit interferenci na přijímači (přijetím překrývajících se posunutých pulzů, způsobené odrazem, pohlcením, ohybem a rozptylem). Imunita je získána díky ignorování stínových impulzů, které nepřekrývají hlavní přijatý impulz v časové doméně (impulzy jsou velmi krátké, v případě posunu o 12 cm vlivem odrazu a rozptylu o délce pulzu 0,4 ns nedojde k překryvu stínovým pulzem). Pro zlepšení těchto vlastností je tedy potřeba zmenšit velikost pulzu a zvýšit zpoždění mezi odesláním aktuálního a nového impulzu, čehož se hojně využívá ve vnitřních prostředích. Obrázek 2.8 ukazuje po po řadě (a) stínový impulz, který nepřekrývá hlavní impulz, (b) stínový impulz, který překrývá hlavní impulz, (c) výsledek překryvu zachycený přijímačem (interference).




Obrázek 2.8: Interference vlivem vícecestného šíření, převzat z [16]

Modulace je proces převádění vstupních dat (série po sobě jdoucích bitů) na reprezentaci vhodnou k odesílání daným médiem. K modulaci UWB signálů může být použita PPM (pulse position modulation), OOK (on-off keying), PAM (pulse amplitudemodulation) a PWM (pulse width modulation), pro další čtení dostupné z [1].

Vzhledem k tomu, že jsou impulzy odesílány periodicky za sebou, vznikají ve frekvenčním spektru špičky (peaky). Ty výrazně omezují maximální výkonové spektrum UWB, které je tím limitováno. Ke zmírnění peaků se používá malý náhodný offset (posun) mezi odesíláním jednotlivých impulzů, který peaky limituje [16].

### Standard IEEE 802.15.4

Tento standard definuje principy v bezdrátových sítích krátkého dosahu WPAN (wireless personal area networks). Skládá se z fyzické a MAC (medium access control) vrstvy, která je součástí vrstvy linkové, referenčního modelu OSI. Fyzická vrstva zajišťuje způsob převodu proudu bitů na signál při vysílání a opačný převod při příjmu. MAC vrstva je subvrstva vrstvy linkové, která zajišťuje přístup k médiu a fyzické adresování [17].

### Standard IEEE 802.15.4a [18]

Jedná se o rozšíření fyzické vrstvy standardu IEEE 802.15.4 o UWB (popsané dále) a CSS (chirp spread spectrum). UWB vrstvě jsou přiřazeny tři rozsahy frekvencí s celkovým počtem 16 kanálů (viz tabulka 2.2).

| Název pásma | Frekvenční rozsah [MHz] | Počet kanálů |
|---|---:|---:|
| Sub-GHz | 24,6-749,6 MHz | 1 |
| Nízké | 3,1-4,8 GHz | 4 |
| Vysoké | 6,0-10,6 GHz | 11 |

Tabulka 2.2: UWB pásma

Dále pak komunikační rychlosti, které jsou: 110 kb/s, 851 kb/s (povinná), 6,81 Mb/s a 27,24 Mb/s. Dle zvolené rychlosti pak získáváme různé pokrytí (10-100 m) a rychlost přenosu dat, dle zamýšlené aplikace. Byla taktéž přidána podpora pro měření vzdálenosti

(TWR – two way ranging, podkapitola 2.1.3). Aby zařízení bylo se standardem kompatibilní, musí být schopno implementovat alespoň jeden kanál v každém pásmu, a to 0, 3 a 9 (viz tabulka frekvenčních pásem 2.3). Ostatní jsou volitelné [19].

| Skupina | Kanál | Středová frekvence [MHz] | Šířka pásma [MHz] | Povinné |
|---|---|---|---|---|
| 0 | 0 | 499,2 | 499,2 | Ano |
| 1 | 1 | 3494,4 | 499,2 | Ne |
| 1 | 2 | 3993,6 | 499,2 | Ne |
| 1 | 3 | 4492,8 | 499,2 | Ano |
| 1 | 4 | 3993,6 | 1331,2 | Ne |
| 2 | 5 | 6489,6 | 499,2 | Ne |
| 2 | 6 | 6988,8 | 499,2 | Ne |
| 2 | 7 | 6489,6 | 1081,6 | Ne |
| 2 | 8 | 7488,0 | 499,2 | Ne |
| 2 | 9 | 7987,2 | 499,2 | Ano |
| 2 | 10 | 8486,4 | 499,2 | Ne |
| 2 | 11 | 7987,2 | 1331,2 | Ne |
| 2 | 12 | 8985,6 | 499,2 | Ne |
| 2 | 13 | 9484,8 | 499,2 | Ne |
| 2 | 14 | 9984,0 | 499,2 | Ne |
| 2 | 15 | 9484,8 | 1354,97 | Ne |

Tabulka 2.3: UWB kanály, jejich frekvence a šířka pásma, dle [18]

UWB rámec je složen z následujících částí [19]:

- Synchronizační hlavička (SHR – synchronization header),
- fyzická hlavička (PHR – PHY header),
- datová část (PSDU).

SHR synchronizační hlavička je dále rozdělena na preamble a začátek rámce (SFD – Start Frame Delimiter), který značí začátek PHY hlavičky. Počet symbolů preamble je volitelný a určuje délku trvání hlavičky – 16, 64, 104, 4096 symbolů, dle aplikace. Delší hlavičky jsou doporučeny pro zlepšení poměru signál-šum (SNR) na přijímačích. Symboly mají délku 31, nebo 127, nabývající hodnot 0, 1 a -1.

Za SHR následuje Fyzická hlavička (PHR), jejíž délka je 16 symbolů. Obsahuje informace pro správné dekódování rámce příjemcem (datová rychlost přenosu dat, délku patičky) a šest paritních bitů.

Poslední položkou jsou data (PSDU), která jsou odeslaná danou rychlostí z možných 0,11, 0,85 (povinné), 6,81, nebo 27,24 Mb/s, obsahující 0-1209 symbolů.

### 2.1.5 Real-time locating system (RTLS)

RTLS jsou systémy, které slouží k lokalizaci, identifikaci a sledování objektů, nebo lidí v reálném čase a v omezeném prostoru (nejčastěji v budově)

Aby bylo možné vývojový tag testovat, bylo potřeba se přizpůsobit technologii UWB lokalizace v zadavatelské společnosti. Tento RTLS systém se skládá ze tří hlavních komponent:

- Tagy (tags) – jsou prvky, které jsou připevněny na sledovaný objekt, ideálně tak, aby jim nic nestínilo v přímém rádiovém dosahu s kotvami.
- Kotvy (anchors) – jsou prvky, které jsou umístěny na statických místech, v ideálním případě tak, aby v každém místě měl tag možnost přímého dosahu ke třem kotvám. Pomocí algoritmu TWR a uživatelské interakce jsou přesně známé pozice kotev i vzdálenosti mezi nimi.
- Server (server) – jedná se o centrální místo, které řídí kotvy a vypočítává přesnou polohu tagů.

V případě využití algoritmu TDOA jsou tagy pravidelnými vysílači UWB signálu, který je zachycen kotvami se známými pozicemi. Pomocí zjištění času příjmu signálu na všech kotvách, které mají synchronizované hodiny, lze spočítat přesnou pozici tagu. Výhoda tohoto způsobu je primárně úspora energie, neboť k vypočítání jedné pozice tagu stačí vyslat pouze jednu zprávu. Vzhledem k tomu, že jsou tagy většinou napájeny z baterie, je nízká spotřeba velice důležité kritérium.

## 2.2 Real-time operační systémy (RTOS)

Vestavěný systém[3] se typicky skládá ze dvou částí. Hardwarovou částí (dále označovanou jako HW) se rozumí sada elektronických komponent a obvodových zapojení. Firmwarová část (dále označovaná jako FW) je řídící program, který implementuje funkcionalitu daného zařízení v mikrokontroléru, který je na desce přítomný a ovládá veškeré periferie.

Firmware lze implementovat několika způsoby. U *bare metal*, je veškerá implementace, spouštění rutin a časování v režii programátora, který má kontrolu nad celým systémem. Z toho důvodu je těžší psát složitější aplikace s mnoha úlohami v krátkém čase a přehledně. Výhodou je poté snadné debugování, neboť kód je lépe predikovatelný (silně záleží na „kráse“ implementace a na tom, co debugujeme). Tento způsob je použit především v jednoduchých, paměťově omezených a časově kritických aplikacích, kde přidání režie v podobě operačního systému by vedlo k větší konzumaci RAM a flash pamětí a delší odezvě. Naproti tomu nevýhodou je náročné řízení v případě mnoha současně běžících podúloh, neboť bare-metal systém běží nejčastěji v nekonečné smyčce (superloop), ve které se úlohy vykonávají sekvenčně. Je pak na vývojářích implementovat funkci paralelizmu.

Další možností je využití operačního systému. Operační systém je zjednodušeně řečeno programové vybavení, které se zavádí ihned po spuštění výpočetního systému a slouží k řízení hardware a poskytnutí jednotného rozhraní aplikacím pro jeho snadné řízení. Nejzákladnější operační systém se skládá ze správy procesů (běžící programy) a správy paměti (pracovní prostor). To, který proces bude aktuálně procesor využívat řeší plánovač (scheduler). Správa paměti se stará především o alokaci a dealokaci paměti. Dále zajišťuje, aby proces nemohl zasahovat do paměti cizího procesu, kam nemá přístup. S operačním systémem také přichází správa uživatelů a knihovny pro snadnou práci s TCP/IP, USB, souborovým systémem, atd. Výhodami použití operačního systému jsou tedy snadnější implementace složitějších aplikací (pokud je programátor znalý) a lepší přehlednost kódu. Nevýhodami jsou především vyšší režie a větší množství využitých RAM a flash pamětí.

Výše uvedené výhody a nevýhody jsou čistě orientační a velmi záleží na konkrétní aplikaci, použitém vybavení a znalosti programátora.

V této kapitole jsou podrobněji popsány klíčové principy RTOS systémů, se zaměřením na nízkou spotřebu. Dále pak shrnutí vlastností některých RTOS systémů s cílem vybrat pro navrhovanou aplikaci ten správný.

### 2.2.1 Klíčové principy RTOS systémů

Real-time application (RTA) je aplikace, u které nezáleží pouze na logické správnosti provedení, ale také na čase, po jakém je výsledek získán. Pokud nejsou dodrženy definované časové požadavky, může nastat kritická chyba (příkladem je airbag, kde je potřeba provést nafouknutí vaku ve vymezené době po nárazu, aby došlo k zmírnění zranění pasažéra. Dále pak stabilizace letadla, ABS systém a další.). Takovéto aplikace sestávají z úloh (procesů, tasků), které mají jasně definovaný termín (deadline), do kterého musí být vykonány.

Real-time OS (operační systém reálného času) je operační systém, který se snaží dodržet časové požadavky takovéto aplikace při současném zachování všech výhod operačního systému, popsaných dříve.

[3] Vestavěný systém (embedded system) je zjednodušeně řečeno výpočetní zařízení, které obsahuje procesor, paměť a veškeré periferie na desce. Takovýto systém slouží k předem stanovenému účelu a je pro něj optimalizován (například digitální hodinky – systém je určen k počítání a zobrazování času, funkce stopek, data a ničeho jiného).

### Task

Celou aplikaci lze rozdělit do pod-úkolů (tasků, synonymně procesů), které se střídají ve využívání procesoru. Můžeme je rozdělit do tří typů podle periody opakování [20]:

- Periodický task – je vykonáván opakovaně s fixně stanoveným časovým intervalem doby vykonávání.
- Aperiodický task – je vykonáván pravidelně, s neznámým časovým intervalem opakování.
- Náhodný (sporadický) task – aperiodický task, který má definován minimální časový interval mezi jednotlivými příchody.

Dále lze task rozdělit podle toho, jaké jsou následky při nedodržení deadlinu[4]:

- Soft task – task, který by měl být vykonán ve svém deadlinu, ale jeho nedodržení nevede k žádným kritickým následkům (například odezva zmáčknutí tlačítka na digitálních hodinkách – při zpoždění se nic neděje).
- Hard task – task, který musí být vykonán ve svém deadlinu, aby nedošlo k možným kritickým následkům (například reakce airbagu na náraz – pokud je provedena později, zvyšuje se riziko zranění pasažéra).

Task může mít dále určenou prioritu, s jakou se má vykonávat. Pro jednoduchost zaveďme konvenci priority 0-7, kde 0 je nejnižší priorita, 7 nejvyšší. Operační systém si u každého tasku ukládá řídící informace do tzv TCB (task control block). Dle různých implementací se může tato struktura lišit. U freeRTOS jsou v ní tyto základní údaje:

- stav procesu;
- ukazatel na vrchol zásobníku, ukazatel na zásobník, hloubka zásobníku (určující pracovní prostor procesu, jeho velikost a aktuální položku);
- priorita tasku (dle strategie plánovače, má vliv na to, s jakou četností bude tasku přiřazen CPU);
- jméno tasku (využito pro debugovací účely).

Stav procesu se dále dělí na:

- Běžící (running) – task aktuálně využívající CPU.
- Připravený (ready) – task připravený na běh, čekající na to, až bude přepnut do stavu running plánovačem.
- Blokovaný (blocked) – task, čekající na určitou událost (delay, semaphore, notification, ...).
- Pozastavený (suspended) – task, pozastaven speciálním příkazem suspend.

Dle toho, jaké tasky se v aplikaci nacházejí, je potřeba přizpůsobit metodu použitého plánování k uspokojení potřeb všech časových požadavků.

[4] Konečný čas, do kterého musí být úkon vykonán

### Správa paměti

Aby task mohl vykonávat svou činnost, musí mít přidělen paměťový pracovní prostor, který může ke svému vykonávání použít. Přidělování paměťového prostoru se děje dvěma způsoby – staticky a dynamicky [21].

Statické přidělení pracovního prostoru probíhá již v době překladu. Nevýhodou je jeho neúplné využití. Výhodou pak přesná znalost využití paměti již v době překladu.

Dynamické přidělování reaguje v reálném čase na aktuální požadavky programu. Může tedy přidělovat a odebírat prostor a tím maximálně využívat paměť. Výzvou je však vznikající fragmentace a možný časový overhead (režie). Různé RTOS systémy využívají různé strategie, v případě freeRTOSu se dají kombinovat.

Řeč zde není pouze o prostoru pro tasky, ale o všech dalších primitivech, které RTOS využívá. O která primitiva se jedná je popsáno v sekci 2.2.1.

### Plánovač (scheduler)

Celý systém lze tedy dekomponovat na jeden nebo několik paralelně nebo sériově běžících tasků. To, který bude aktuálně běžet a na jak dlouho, zajišťuje scheduler. V případě konvenčních operačních systémů se zaměřuje především na využití CPU [%] (spravedlnost rovnoměrného využití procesoru všemi procesy), systémovou propustnost (počet dokončených tasků za čas) a minimalizaci odezvy. V případě RTOS se musí jednat především o dodržení všech časových požadavků. To, jaký plánovač vybrat, záleží pouze na návrháři a na aplikaci, kterou implementuje.

Preemptivní plánování je takové plánování, při němž OS může násilně odebrat procesor procesu. Nepreemptivní (kooperativní) plánování je takové, kdy lze odebrat procesor procesu pouze tím způsobem, že se jej proces sám vzdá (yield). U preemptivního plánování dochází nejčastěji ke spuštění časovače (například sysTick u ARM Cortex M3), který periodicky volá přerušení, které spustí výběr následujícího tasku určeného k běhu a případně proběhne přepnutí kontextu. Plánování lze obecně rozdělit na dvě kategorie [22]:

- Offline – plán přepínání kontextu je naplánován ještě před samotným spuštěním aplikace. Je potřeba znát všechny atributy systému (deadliny, doby vykonávání, doby startu). Jejich výhodou je minimální režie.
- Online – rozhodnutí o výběru následujícího procesu se řeší za běhu. Jejich výhoda je přizpůsobení se neočekávaným situacím.

Následuje výčet základních plánovacích politik, které přiřazují procesor připraveným taskům:

- First come, first served (FCFS, FIFO) – je nepreemptivní politika, kde jsou procesy obsluhovány v pořadí, v jakém přišly. Běží tak dlouho, dokud se nevzdají CPU. Nevýhodné pro krátké procesy. Efektivní z hlediska minimalizace přepínání kontextu a jednoduchosti implementace.
- Round robin – je preemptivní politika, kde má každý proces přiděleno kvantum času pro svůj běh. Pokud za tu dobu nestihne dokončit svou činnost, musí počkat na další kolo, až také ostatní procesy využijí své kvantum, nebo se vzdají procesoru předčasně. Tato politika je spravedlivá pro všechny procesy s rychlou odezvou. Nevýhodou je režie přepínání. Určení velikosti kvanta je netriviální (režie vs. odezva).

- Shortest job first (SJF) – procesor získá ten proces s nejkratší dobou vykonávání.
- Priority driven scheduling – procesor je přiřazen vždy tasku s nejvyšší prioritou. Nevýhodou je hladovění tasků s nižší prioritou, což může být vykompenzováno dynamickým přidělováním priorit.
- Rate-monotonic scheduling (RMS) – jedná se o statické plánování. Priorita je úlohám přidělována podle jejich periody. Čím kratší je perioda úlohy, tím větší je její priorita.
- Deadline-monotonic scheduling (DMS) – taskům jsou přiřazeny priority podle jejich deadlinů. Čím kratší je deadline, tím vyšší je priorita.
- Earliest deadline first (EDF) – taskům jsou dynamicky za běhu přidělovány priority podle toho, kterému z nich nejdříve vyprší deadline (je potřeba dokončit task co nejrychleji, aby byl splněn časový požadavek).

#### Úskalí více-procesového systému

V případě systému, ve kterém dochází k současnému běhu několika střídajících se procesů, může dojít k událostem, které zapříčiňují nesprávný nebo neočekávaný běh programů a špatně se debugují:

- Deadlock – stav, kdy ve skupině procesů každý z nich čeká na provedení akce (vzdání se prostředku, přijetí zprávy) jiným členem. Dochází k zablokování procesů.
  - Řešením může být zamezení jedné ze čtyř Coffmanových podmínek. Detekce probíhá sledováním stavu a alokováním zdrojů procesů.
- Priority inversion – nastává tehdy, pokud nízko-prioritní proces blokuje běh procesu s vyšší prioritou. Blokování probíhá prostřednictvím vlastnění prostředku, ke kterému má aktuálně nízko-prioritní proces výlučný přístup.
  - Priority inheritance – řešení, kdy proces s nižší prioritou dočasně zdědí prioritu čekajícího procesu s nejvyšší prioritou.
  - Priority ceiliing – řešení, kdy jsou všem zdrojům přiřazeny priority, které se rovnají nejvyšší prioritě všech tasků, které si je mohou získat.
- Livelock – obdoba deadlocku, kdy procesy místo čekání provádí jiný kód, který neustále vytěžuje CPU.
- Starvation (hladovění) – nastává, když se proces po dlouhou dobu nedostává k požadovaným zdrojům.
  - Řešením může být výběr správného plánovacího algoritmu.
- Race condition – vzniká, pokud procesy používají stejný prostředek (paměť, sdílená proměnná) zároveň (například, procesy tisknou řetězec na stejný výstup, kvůli přepnutí kontextu se výstupy promíchají).
  - Řešením je zaobalení kódu, který využívá daný prostředek, do kritické sekce. Ta může být implementována pomocí mutexů a binárních semaforů. Další možností je pak vypnutí přerušení, což znemožní přepnutí kontextu a vstup jiného procesu do sekce, nebo provedením operace atomicky.

#### Prostředky

Operační systémy umožňují použití primitiv, která umožňují vzájemnou výměnu dat mezi procesy, jejich synchronizaci, bezpečný přístup ke zdrojům, uspávání a další, ulehčující vývoj programátorům [23]:

- semafor – mimo ochrany kritické sekce umožňuje signalizaci mezi procesy. Základem je numerická hodnota, která se s příchodem procesů snižuje. Pokud je menší-rovna nule, proces čeká. Pokud je hodnota větší, proces pokračuje. Umožňuje projití několika procesů zároveň;
- bninární semafor – stejný princip jako u semaforu s tím rozdílem, že jsou hodnoty binární. Umožňuje projití jen jednoho procesu;
- mutex – objekt, který implementuje výlučný přístup v kritické sekci pomocí zamykání a odemykání pomyslného zámku;
- fronta – předávání dat různé délky mezi procesy, FIFO;
- mailbox – předávání fixních dat mezi procesy, FIFO;
- práce s časem – časovače, zpoždění, odměření času.

### 2.2.2 Techniky nízké spotřeby

Existuje několik metod, jakými lze dosáhnout nízké spotřeby u vestavěných systémů z pohledu firmware. Jedná se například o[24]:

- vypnutí periferií, které nepoužíváme;
- snížení frekvence a napětí (dynamic voltage and frequency scaling);
- napájecí režimy, do kterých můžeme přepnout periferie, které zrovna nepoužíváme, nebo je používáme méně často (low power mode, sleep mode, deep sleep).

Následuje výčet technik, které lze použít v RTOS systému.

#### IDLE hook

U většiny RTOS systémů (například FreeRTOS) je spuštěn speciální IDLE task s nejnižší prioritou, který běží pouze tehdy, nemá-li systém nic na práci. Je spuštěn pro případ, že by nebyl v systému definován žádný jiný task k běhu. Mimo jiné se stará o uvolňování paměti. Lze ručně doprogramovat přepnutí procesoru do nízko-příkonového režimu v případě, že se tento task dostane ke slovu. Nevýhodou je ovšem periodické probouzení CPU kvůli spouštění preemptivního časovače. Na zvážení tedy zůstává, zda se přechod do spánku a následné probuzení vyplatí [21].

#### Tickless mode

Tato technika, využívaná v RTOS systémech, využívá znalosti plánovače, který ví, za jak dlouho je naplánováno spuštění dalšího procesu. Díky této znalosti dokáže přepnout MCU do režimu spánku na dobu nezbytnou ke spuštění tohoto procesu.

Probuzení pak probíhá pomocí přerušení z předem nastaveného časovače. Procesor může také vzbudit jakákoliv jiná událost přerušení (záleží dle konkrétní platformy), například GPIO, nebo jiná periferie.

### 2.2.3 Srovnání RTOS systémů

Zkoumanými vlastnostmi pro srovnání RTOS systémů jsou: podpora ARM Cortex M4 architektury (plynoucí ze zadání), techniky nízké spotřeby, licence (cena), předcházení race conditions, spotřeba RAM a flash pamětí, průměrné časy, typy plánovačů a subjektivní pocit z dostupnosti materiálů a uživatelské komunity.

Tato analýza je velice okrajová. Důraz je kladen především na první tři aspekty, tedy podpora ARM Cortex M4, licenci a nízkou spotřebu. Kolonky paměti a časů je potřeba brát s rezervou, neboť testy byly prováděny na rozdílných platformách s využitím jiných nastavení systémů.

#### FreeRTOS (verze 9.0.0, 10.0.0) [25][21]

Jedná se o operační systém reálného času se začátkem vývoje před patnácti lety, psán převážně v jazycích C a Assembleru. Podporuje více než čtyřicet architektur, mezi kterými je i ARM Cortex M4. Jeho vlastnostmi jsou:

- Licence, cena:
  - Tento systém, distribuovaný pod MIT licencí, je možno zdarma libovolně používat, modifikovat, prodávat, ovšem bez jakékoliv záruky korektního fungování. K tomu, aby bylo možné získat komerční verzi s podporou, nebo dokonce certifikovanou verzi (IEC 61508 SIL 3, IEC 62304, FDA 510(K), ISO 26262), je potřeba si koupit openRTOS, nebo safeRTOS, které sdílí stejné jádro, jako freeRTOS, ovšem s dalšími garancemi.
- Nízká spotřeba:
  - Podpora IDLE hook i tickless režimu (tickless režim prozatím nedostupný pro všechny podporované MCU).
- Race conditions:
  - Předcházení priority inversion pomocí mutexu, který používá algoritmus priority inheritance.
- Plánovač:
  - Možno použít preemptivní i kooperativní verzi prioritního plánovače, který dává vždy přednost tasku s nejvyšší prioritou. Pokud mají tasky stejnou prioritu, probíhá jejich střídání pomocí algoritmu round-robin.
- Paměť (orientačně):
  - RAM – statické i dynamické ukládání objektů (dynamických 5 úrovní, s různými strategiemi a řešením fragmentace);
  - RAM – plánovač (236 B), Fronta (76 B + místo na uložení dat), Task (64 B, včetně jména procesu 4 B + velikost zásobníku), semaphore, mutex (nepřesné informace z webových fór cca 100 B a výše);
  - flash – kernel s velikostí 5-10 kB.
- Čas (orientačně):

  - doba přepnutí kontextu je silně závislá na mnoha faktorech. Udává se průměrně 84 cyklů;
  - doba startu systému, včetně zapnutí plánovače je cca 1200 CPU cyklů. Vytvoření fronty, semaforu a mutexu 500 cyklů. Vytvoření tasku 1100 cyklů.

- Další:
  - FreeRTOS má bohatou podporu v oblasti Q&A, mnoho příkladů a čitelný, dobře strukturovaný manuál;
  - porpora kritických sekcí, časovačů, uspávání tasků, front, mutexů, semaforů.


Obrázek 2.9: Logo freeRTOS, převzato z [25]

**embOS (verze 5.8.2)** [26][23]

Jedná se o Operační systém reálného času (25 let vývoje), od společnosti SEGGER Microcontroller, psaný v ANSI C a Assembleru. Podporuje přes 30 architektur, mezi nimiž je i ARM Cortex M4.

- Licence, cena:
  - Tento systém je zdarma pro jakékoliv nekomerční, či edukační účely. Pro komerční účely je zpoplatněn a nabízí se v několika verzích. Základní verze nabízí certifikace IEC 61508 SIL 3 a IEC 62304 Class C. Je kompatibilní s MISRA-C 2012. Verze embOS-Safe zahrnuje předchozí verzi a nabízí možnost snadného získání certifikace ISO 26262.

- Nízká spotřeba:
  - Podpora IDLE hook i tickless režimu. Jednodušší práce s tickless režimem.

- Race conditions:
  - Předcházení priority inversion pomocí mutexu, který používá algoritmus priority inheritance.

- Plánovač:
  - Možno použít preemptivní i nepreemptivní verzi prioritního plánovače, který dává vždy přednost tasku s nejvyšší prioritou. Pokud mají tasky stejnou prioritu, probíhá jejich střídání pomocí algoritmu round-robin.

- Paměť (orientačně):
  - statické i dynamické ukládání objektů;

  - RAM – kernel (71 B), semafor (8 B), mutex (16 B), časovač (20 B);
  - flash – kernel 1700 B.

- Další:
  - podpora na Q&A v podobě fóra, srozumitelný manuál s příklady;
  - podpora watchdogu, který kontroluje, zda tasky, SW časovače a interrupty nejsou zaseknuté;
  - podpora kritických sekcí, časovačů, uspávání tasků, front, mutexů, semaforů, mailboxů.


Obrázek 2.10: logo embOS, převzato z [26]

### Micrium uC/OS-III (verze 5.8.2) [26][23]

Operační systémy uC/OS-X, reálného času, byly poprvé představeny v roce 1992. Verze uC/OS-III, představena v roce 2009 je přímým nástupcem uC/OS-II. Je psaný v ANSI C a assembleru a podporuje přes 30 architektur, mezi nimiž je i ARM Cortex M4. Certifikace je prozatím ve vývoji.

- Licence, cena:
  - Tento systém je nabízen zdarma v podobě dostupných zdrojových kódů pro jakékoliv nekomerční, či edukační účely. Pro komerční účely je však zpoplatněn. Je kompatibilní s MISRA-C 2004, vyjma sedmi pravidel. Verze nabízí možnost bezpečnostních certifikací.
- Nízká spotřeba:
  - Podpora IDLE hook režimu.
- Race conditions:
  - Předcházení priority inversion pomocí mutexu, který používá algoritmus priority inheritance
- Plánovač:

  - Možno použít preemptivní verzi prioritního plánovače, který dává vždy přednost tasku s nejvyšší prioritou. Pokud mají tasky stejnou prioritu, probíhá jejich střídání pomocí round-robin (nemožné ve verzi uC-OS II).

- Paměť (orientačně):
  - statické i dynamické ukládání objektů;
  - využití kódového segmentu 6-24 kB;
  - využití více, než 1 kB RAM paměti.

- Další:
  - srozumitelný manuál s příklady;
  - měření výpočetních časů a využití paměti všech procesů, předcházení deadlockům použitím časovačů uspaných tasků;
  - podpora časovačů, front, mutexů, semaforů, signálů, kritických sekcí.

### 2.2.4 Výběr RTOS systému

Všechny zde popsané systémy splňují požadované vlastnosti: podpora Cortex M4 architektury, režim nízké spotřeby, předcházení race conditions, stejné režimy plánovačů, dobrá podpora výrobce v podobě manuálů s příklady.

Zásadním rozdílem je však licence. FreeRTOS jako jediný umožňuje použití i v komerčních projektech zdarma a nabízí možnost přejít na placenou verzi s dalšími garancemi v podobě podpory a certifikací, při současném zachování kompatibility. Vzhledem k této vlastnosti byl pro další implementaci vybrán systém FreeRTOS.

## 2.3 Shrnutí

V této kapitole byly popsány principy bezdrátové lokalizace ve vnitřních a venkovních prostředích. Popsány byly technologie ve vnitřních prostředích – technologie rádiové, akustické a senzorové. Důraz byl kladen především na technologie rádiové. Popis byl následován výčtem metod získání přesné pozice objektů. Detailně byl proveden popis technologie UWB, s důrazem na standard IEEE 802.15.4a.

V další podkapitole byly detailně popsány klíčové principy RTOS systémů. Byli vybráni tři kandidáti a z nich na základě specifikovaných požadavků vybrán systém FreeRTOS. Byl vybrán primárně z důvodu možnosti jeho bezplatného užití i v komerčních projektech s možností přechodu na lepší verzi se zachováním zpětné kompatibility.

# Kapitola 3

# Návrh a realizace HW

Tato sekce je zaměřena na návrh a výrobu HW platformy tagu. Popisuje specifikaci, na kterou bylo potřeba brát při návrhu zřetel, výběr technologie, postup výběru součástek pro cílovou platformu splňujících danou funkcionalitu a vlastnosti a postup návrhu samotného schématu i s layoutem. Na závěr kapitoly pak výrobu desky i s oživením.

K návrhu obvodu byl použit nástroj Eagle, verze 9.5.1, společnosti Autodesk [27]. Eagle je nástroj sloužící pro návrh desek plošných spojů (DPS). Umožňuje především vytváření schémat, rozložení a propojení součástek na desce, automatické nebo ruční routování propojů, správu a vytváření knihoven součástek, elektrickou a návrhovou kontrolu obvodu a mnohé další. Tento nástroj byl zvolen pro jeho širokou škálu funkcí, jednoduchost použití a také díky poskytované studentské licenci, která umožňuje funkcionalitu profesionální verze.

## 3.1 Specifikace požadavků

Při návrhu schématu bylo třeba vzít v úvahu specifikaci. Tuto specifikaci lze rozdělit do tří kategorií: dle zadání diplomové práce, volitelných dodatečných rozšíření/vylepšení a odvozených požadavků nepřímo plynoucích ze zadání.

Tabulka 3.1 popisuje specifikaci HW komponent plynoucí ze zadání.

Dále následují volitelné specifikace, které jsou rozšířením práce, neboť nebyly součástí zadání. Zachycuje je tabulka 3.2.

Posledními specifikovanými jsou součástky, nezbytné pro doplnění celkové (nezbytné) funkcionality tagu, zachycené v tabulce 3.3.

### 3.1.1 Blokový diagram obvodu

Dle předchozího výčtu součástek následuje sestavení předběžného blokového diagramu obvodu, k nahlédnutí na obrázku 3.1. Jelikož nejsou přesně známy detaily součástek, nelze diagram modelovat s většími detaily, jako jsou přesná propojení a použité sběrnice. Slouží však jako zjednodušený všeobecný přehled.

| Typ periferie | Popis |
|---|---|
| UWB rádiový modul | Slouží pro vysílání a přijímání UWB rádiových zpráv. Tento modul by měl splňovat specifikaci dle IEEE 802.15.4a. Primárně má sloužit k vysílání UWB zpráv kotvám, pomocí nichž se zjišťuje poloha tagu. Modul by měl fungovat také jako UWB přijímač v případě možného příjmu konfigurace, nebo jiných dat. |
| Mikrokontrolér ARM Cortex M4 | Slouží pro implementaci veškeré funkcionality. Důraz by měl být kladen na nízkou spotřebu. Dále pak na dostatečnou velikost RAM a flash pamětí, kvůli nutnosti uložení jádra operačního systému a jeho provozu. Především musí být založen na architektuře ARM Cortex M4. |
| Akcelerometer | Slouží k detekci pohybu tagu. Měl by umožňovat podporou přerušení při přesažení stanoveného prahu akcelerace. |
| E-ink displej | Slouží k zobrazování provozních a uživatelských informací. |

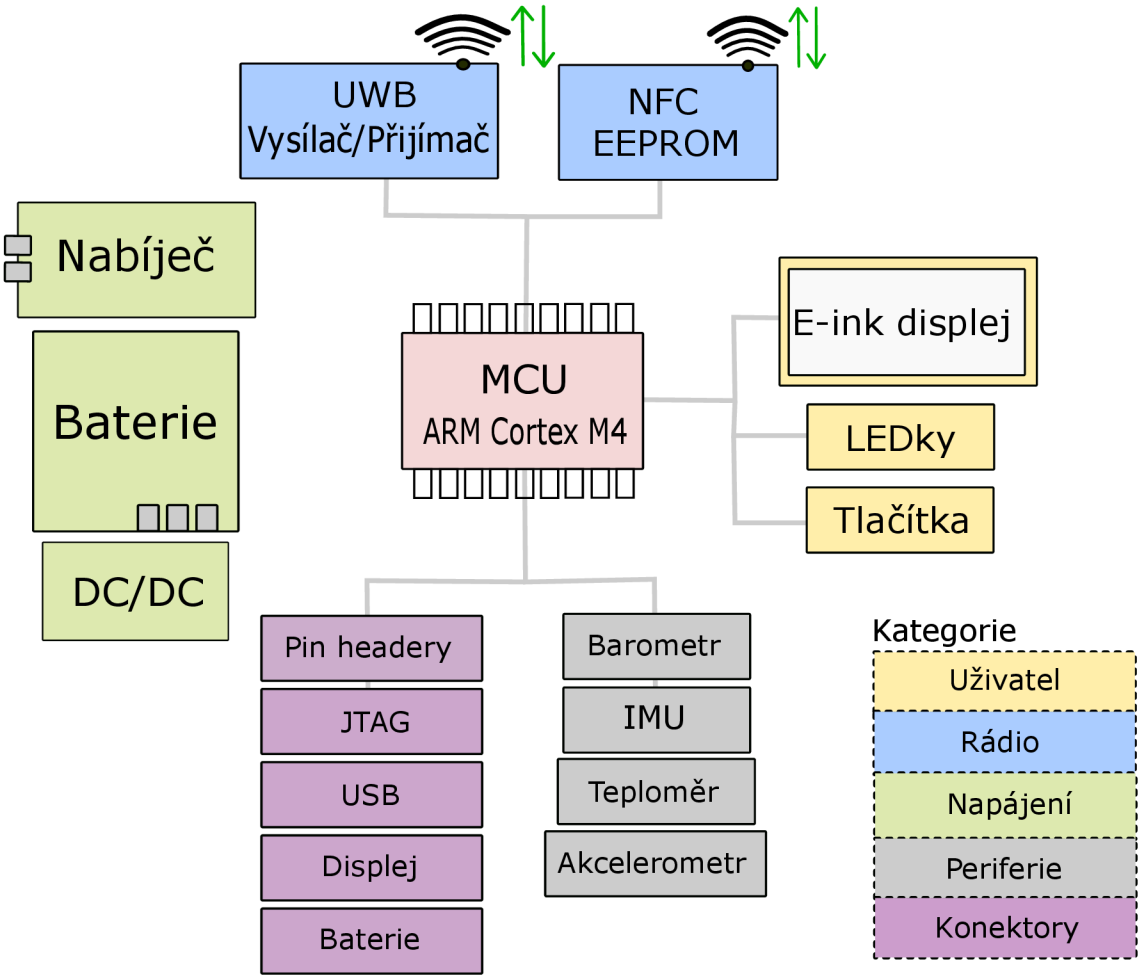
Tabulka 3.1: Specifikace součástek plynoucí ze zadání

| Typ periferie | Popis |
|---|---|
| IMU jednotka | Inerciální jednotka by měla sloužit k určení natočení tagu v 3D prostoru. Tyto informace poslouží uživateli k analýze údajů pohybujícího se objektu. |
| Barometr | Slouží k upřesnění polohy tagu, primárně pak jeho výšky. |
| NFC EEPROM | EEPROM paměť, která slouží k uložení a vyčtení uživatelských informací (nastavení) pomocí NFC technologie. |
| Teplotní čidlo | Slouží k analýze teploty pohybujícího se objektu a analýze provozního stavu tagu. |
| Tlačítka | Slouží k ovládání tagu. |
| Diody | Slouží k indikaci provozních informací. |
| Baterie | Umožňuje použití tagu v reálném prostředí a jeho snadnější debugování. |

Tabulka 3.2: Specifikace součástek k rozšíření práce

| Typ periferie | Popis |
|---|---|
| Nabíjecí obvod (charging controller) | Zajišťuje nabíjení baterie (ideálně i napájení obvodu v případě nepřítomnosti baterie). |
| DC-DC měniče | Dodává přesně stanovené napětí obvodům. |
| Konektory | JTAG (pro debugování), pin headery (připojení dalších periferií, usnadnění měření), konektor pro displej a baterii. |
| Podpůrné součástky | Rezistory, kondenzátory, cívky, diody, ... |

Tabulka 3.3: Odvozené specifikace




Obrázek 3.1: Předběžný blokový diagram obvodu

### 3.1.2 Technologie

Před samotným výběrem součástek je potřeba si rozmyslet výrobní technologii, která bude při výrobě desky použita. Technologie ovlivňuje provedení součástek – jejich pouzdra, odolnost, délku a provedení pinů.

Byla zvolena technologie SMT (Surface mount technology) pro její výhody oproti THT (Through-Hole technology). Výslednými vlastnosti jsou: menší velikost součástek a tedy velikost výsledného obvodu, možnost umístit součástky na obě strany PCB, menší parazitní kapacita součástek, více místa při routování (vytváření vodivých propojení mezi součástkami).

Pro pájení byla vybrána technologie pájení přetavením, jelikož na fakultě bylo dostupné potřebné vybavení, což umožnilo manuální výrobu. Výběr této technologie vedl také ke zjednodušení celého procesu návrhu, jelikož oproti technologii pájení vlnou odpadla nutnost dodržení minimálních vzdáleností a natočení součástek.

## 3.2 Výběr a popis součástek

Popsané technologie v předchozí podkapitole nejsou to jediné, na co je potřeba brát zřetel. Zřetel je dále potřeba brát na výběr napájecího napětí, spotřebu, elektromagnetickou charakteristiku součástek, kompatibilitu sběrnic a přenosové rychlosti. Součástky lze seřadit podle důležitosti následovně:

1. UWB modul – jelikož jich není na trhu mnoho, odvíjí se od něj zbytek výběru.
2. MCU – druhá nejdůležitější komponenta, která nabízí různé parametry. Musí vyhovovat účelu aplikace, například: nízká spotřeba, periferie, velikost paměti, výpočetní výkon, dobrá podpora výrobce, dostupné knihovny...
3. Další periferie – veškeré dále jmenované periferie jsou dostupné u mnoha výrobců v mnohých provedeních. Jejich výběr je tedy přizpůsoben výběru MCU a UWB modulu.

### 3.2.1 Bezdrátový UWB transceiver

K říjnu roku 2019 byli na trhu zmapováni tři výrobci UWB čipů:

- Apple – čip U1, osazen v Apple iPhone 11 pro účely lokalizace, funkcionálně podobný DW1000, v době psaní této práce nedostupný [28].
- NXP – prodej zahájen, produkce však zahájena v polovině roku 2020 [29].
- DecaWave – světově první čip, který podporuje UWB (první zmínky okolo roku 2013). Na trhu má prozatím společnost Decawave v oblasti UWB jednoznačný monopol a nejlepší dostupnost [30].

Společnost DecaWave nabízí více variant UWB čipu [31]:

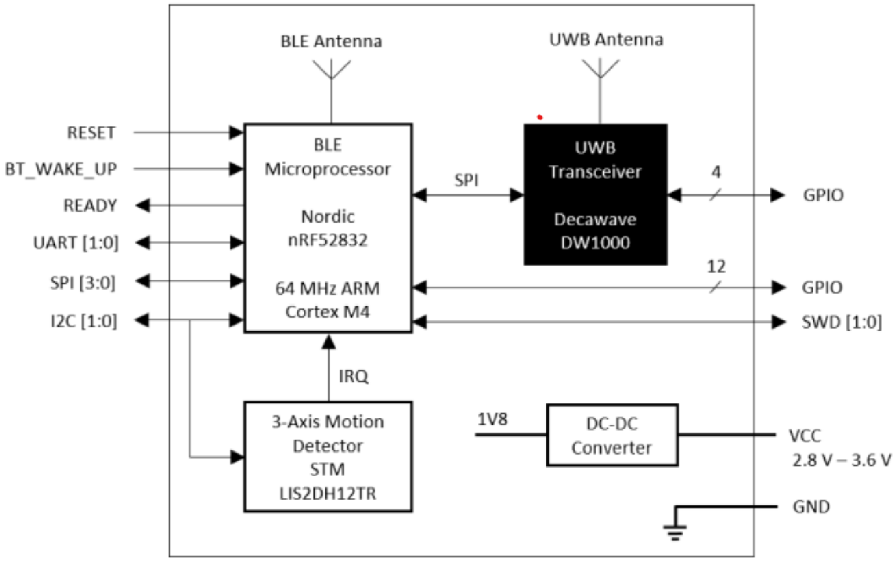
- DW1000 – samostatný UWB transceiver, který komunikuje přes SPI a je kompatibilní se standardem IEEE802.15.4-2011. Neobsahuje anténu ani žádné další periferie.
- DWM1000 – modul, který obsahuje UWB transceiver DW1000, anténu a přídavnou elektroniku, komunikující přes SPI.
- DWM1001 – modul, který obsahuje UWB transceiver DW1000, anténu a mikrokontrolér Nordic Semiconductor nRF52832. Tento mikrokontrolér je založen na ARM Cortex M4 jádru, obsahuje Bluetooth a akcelerometr. Spotřeba ve sleep módu je udávána $< 5$ μA.

Byl vybrán modul DWM1001 (viz schéma 3.2 a ukázka 3.3), protože obsahuje anténu, akcelerometr, mikrokontrolér s požadovanou architekturou a má nízkou spotřebu. Od vlastností tohoto modulu se bude odvíjet výběr dalších součástek. Výběr mikrokontroléru a akcelerometru je tedy také hotový, neboť se součástky nachází v samotném modulu.

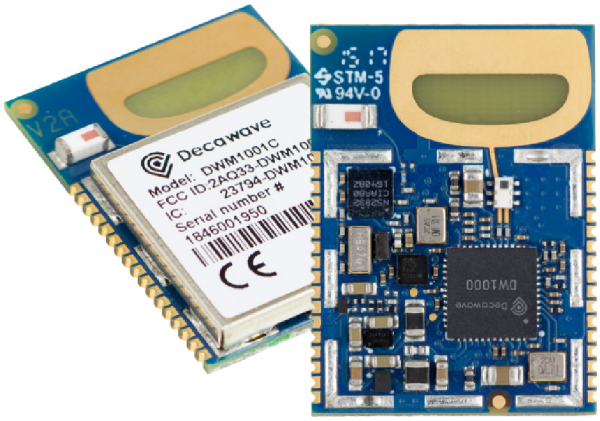
Modul DWM1001 disponuje následujícími vlastnostmi:

- Obsahuje DW1000 UWB transceiver, UWB PCB anténu (6.5 GHz středová frekvence), ARM Cortex M4F – Nordic Semiconductor nRF52832, Bluetooth, 3-osý akcelerometr STM LIS2DH12TR.
- Spotřeba ve sleep módu je <5 μA.

- Napájecí napětí je 2.8–3.6 V.
- Podporuje rozhraní SPI, $I^2C$, UART, NFC, GPIO, SWD a další.




Obrázek 3.2: Schéma modulu DWM1001, dostupné z [31]




Obrázek 3.3: Modul DWM1001, dostupné z [31]

### DW1000 UWB transceiver – popis

DW1000 komunikuje po sběrnici SPI, maximálně však 20 MHz. Jeho spotřeba je 31 mA při vysílání, 64 mA při příjmu. V režimu deep sleep pak 100 nA. Je dodáván v malém 6 × 6 mm QFN pouzdře. Podporuje protokol IEEE 802.15.4-2011. Umožňuje lokalizaci s 10 cm přesností a dosah 30 m. Umožňuje měření vzdálenosti dle algoritmů TWR, dále TOF a TDOA. Umožňuje délku paketu až 1023 B. Modulace probíhá pomocí BPM s BPSK.

### 3.2.2 Mikrokontrolér

Mikrokontrolérem je dle předchozího výběru Nordic Semiconductor nRF52832_xxAA. Tento mikrokontrolér je založen na architektuře ARM Cortex M4F. Tato architektura obsahuje SysTick timer, DSP jednotku, floating-point jednotku a dva ukazatele na zásobníky (PSP a MSP, které podporují řízení operačním systémem). Disponuje 64 kB RAM a 512 kB flash paměti. Pracuje na frekvenci 64 MHz.

Dále obsahuje periferie pro Bluetooth, NFC, GPIO s interními pull-up/pull-down rezistory, další periferie ($I^2C$, SPI, UART) – mapovatelné na jakékoliv piny, 128b AES koprocesor, RAM retention (zachování obsahu RAM ve sleep módu), ADC a další. Jeho spotřeba je 51,6 µA. V úsporném režimu se zachováním RAM paměti a probouzením na jakoukoliv událost pak 1,5 µA. Podporuje přerušení hranou i úrovní. Pozor je třeba si dát na GPIO piny (při návrhu schématu), které se nachází poblíž pinů antény, u kterých se nedoporučuje, aby jimi procházely vysokofrekvenční signály. Pro debugování podporuje technologie SWD, DWT, ETM a ITM.

### 3.2.3 Další součástky

Následující sekce popisuje další součástky takovým způsobem, aby bylo možné navrhnout hardware i strukturu firmware. Řeč je hlavně o rozhraní, počtu GPIO vodičů (primárně přerušení) a typu pouzder. Výběr veškerých součástek je zaměřen především na nízkou spotřebu, cenu, velikost a napájecí napětí 3,3 V, dle napájecího napětí MCU. Tento fakt dále nebude zmiňován.

#### Akcelerometr

Akcelerometr LIS2DH12TR od společnosti STM je již obsažen v modulu DWM1001. Komunikuje přes rozhraní $I^2C$ (*SDA, SCL*) a jeden vodič přerušení (*INT*). Umožňuje nastavení prahové hodnoty, při jejímž překročení dojde k přerušení (detekce pohybu). Jeho spotřeba může být snížena až na 2 µA.

#### Barometr

DPS310 od společnosti Infineon, je barometr, který komunikuje přes rozhraní $I^2C$ (*SDA, SCL*). Obsahuje také teplotní senzor. Spotřeba při rychlosti 1 Hz je 1,7 µA pro tlaková data. Vyráběn je v pouzdře LGA, velikosti 2,0 × 2,5 mm.

#### IMU jedotky

BMX160 od společnosti Bosch, obsahuje magnetometr, gyroskop i akcelerometr. Komunikuje přes rozhraní $I^2C$ se dvěma přerušeními. Spotřeba se pohybuje v rozmezí 5 µA (suspended)-1585 µA (full operation). Vyráběn v pouzdře LGA, v rozměrech 2,5 × 3,0 mm.

Návrhem zadavatelské společnosti bylo přidání další IMU jednotky pro účely testování a možného budoucího zapracování do dalších produktů. Jedná se o dva samostatné čipy, umožňující dodání funkcionality 9D IMU jednotky. Jedná se o akcelerometr a gyroskop s označením ISM330DLC v jednom čipu, který komunikuje přes $I^2C$ (*SDA, SCL*) a dva vodiče přerušení (*INT1, INT2*). Typ pouzdra je LGA, ve velikosti 2,5 × 3 mm. Jeho spotřeba se pohybuje v rozmezí 0,35-0,75 mA při fungování obou senzorů a záleží na frekvenci snímání. V power-down režimu konzumuje 10 µA.

### Magnetometr

Magnetometrem je senzor IIS2MDC od společnosti ST, který doplňuje předchozí ISM330DLC o magnetická data. Komunikuje pomocí $I^2C$ (*SDA, SCL*) a vodiče přerušení (*INT*). Vyráběn v pouzdře LGA, s rozměry 2,0 × 2,0 mm, se spotřebou v rozmezí 1,5–1130 µA dle vybraného módu.

### E-ink displej

Waveshare 2.13 inch e-Paper [32] je černobílý, 2.13 palcový E-ink displej s rozlišením 122 × 255 px, se spotřebou 26,4 mW (aktualizace displeje) a 0,017 mW (standby). Komunikuje přes nestandardní SPI rozhraní. Toto rozhraní využívá pouze jeden datový vodič a je dále popsáno v kapitole konstrukce ovladačů k firmware. Displej je připojen pomocí 24pinového PFC24/0,5 mm konektoru. Jednotlivé signály jsou:

- *DIN* – MOSI výstup na SPI řídícího mikrokontroléru;
- *CLK* – časování SPI (synchronní);
- *CS* – SPI výběr zařízení slave (aktivní v L);
- *DC* – datový/příkazový kontrolní pin (H pro data, L pro příkaz);
- *RST* – externí reset (aktivní v L);
- *BUSY* – indikace zaneprázdnění (aktivní v L);
- *BS* – výběr režimu nestandardního SPI.

### NFC EEPROM

Výběr tohoto čipu včetně implementace knihovny a návrhu zapojení byl proveden zadavatelskou firmou. Jedná se o čip NT3H2X11, od společnosti NXP. Slouží jako úschovna uživatelských nastavení, která umožňuje jejich přepis přes NFC i v nepřítomnosti interního napájení. Je připojen pomocí $I^2C$ (*SDA, SCL*) a pinu pro detekci NFC pole (*FD*), který slouží pro detekci aktualizací nebo přečtení údajů.

### Konektory

Pro debugování byl zvolen 10pinový Cortex-debug konektor, s roztečí 2,54 mm. Tento konektor obsahuje piny:

- *SWDIO* – vstupně/výstupní pin pro přenos dat (SWD);
- *TCK* – časování k *SWDIO* pinu (SWD);
- *RESET* – resetování MCU;
- *VREF* – referenční napětí obvodu;
- *GND* – vyvedení společné země.

Dalšími konektory jsou pin headery, které umožňují rozšíření funkcionality výsledného modulu a testovací měření. Pro připojení displeje je, dle datasheetu výrobce, použit konektor PFC24/0,5 mm. K připojení napájení byl vybrán USB Mini-B konektor.

#### Napájení

Aby bylo možné provádět testování vývojové desky tagu v terénu, bylo nezbytné zakomponovat k tagu jednočlánkovou Li-Pol baterii. Konkrétně se jedná o LP-402933-1S-3. Má kapacitu 300 mAh s rychlostí nabíjení 150 mA. Teoreticky lze tedy baterii nabít za 2 hodiny. Nominální napětí je 3,7 V, což je pro napájení obvodu pracujícího na 3,3 V dostatečné. Baterie taktéž disponuje pinem termistoru k hlídání její teploty a obsahuje interní ochranu.

Jako nabíjecí obvod byl využit obvod LM3658 od Texas instruments. Umožňuje napájení obvodu i při vyndané baterii (LDO mód). Vstupem obvodu je napětí z nabíječky, nebo USB portu počítače, o napětí 4,5-6 V. Z důvodu použití nabíjení i z USB portu je nutné omezit nabíjecí proud obvodu na 100 mA (k odebírání 500 mA z USB 2.0 je zapotřebí enumerace a povolení od hostitelského zařízení, což by přidalo další obvod a zkomplikovalo zapojení). HW omezením čipu (přidání vhodného rezistoru k pinu nabíječe) je tedy nastaven maximální odběr 100 mA. Obvod indikuje svou činnost pomocí dvou výstupů (*STAT1, STAT2*), které jsou připojeny k indikačním diodám a možností vstupu do MCU.

Výstupní napětí nabíječe a baterie je potřeba dále regulovat na 3,3 V. To je zajištěno DC-DC regulátorem TPS7A05, společnosti Texas Instruments. Umožňuje dodávat výstup až 200 mA.

#### LEDky, tlačítka, přepínače

Byly přidány také dvě LEDky (LED4, LED5), které indikují stav nabíjení, dvě LEDky (LED1, LED2) určené k uživatelskému použití a jedna LEDka (LED3) pro debugování.

Pro uživatelský vstup slouží dvě tlačítka SPST (S2, S3) a dále jedno tlačítko SPST (S1) k resetování MCU.

V obvodu jsou dále zakomponovány čtyři přepínače SPDT, které slouží k zapnutí napájení obvodu (S4), výběru SPI módu displeje (S6), přepnutí typu použitého displeje (S5) a k uživatelskému použití (S4).

#### Další součástky

Vzhledem k požadovaným malým rozměrům byla pro většinu dalších součástek použita pouzdra velikosti především 0603 (1,6 × 0,8 mm), technologie SMT.

## 3.3 Návrh schématu zapojení lokalizačního modulu

Jak bylo zmíněno v úvodu kapitoly, výsledné schéma bylo modelováno za použití nástroje Eagle CAD.

### 3.3.1 Řídící obvody

Každá z daných součástek ve svém datasheetu nebo aplikačních poznámkách obsahuje i doporučený způsob zapojení. Použitím těchto doporučení vznikl výsledný obvod. Obsahuje filtrační kondenzátory a pull-up rezistory doporučených hodnot. MCU nabízí možnost nastavení interních pull-up, nebo pull-down rezistorů na všech GPIO pinech. Díky tomu jich nebylo potřeba u SWD vodičů pro debugování ani u uživatelských tlačítek a *FD* pinu indikujícího detekci NFC pole.

### 3.3.2 Knihovny součástek

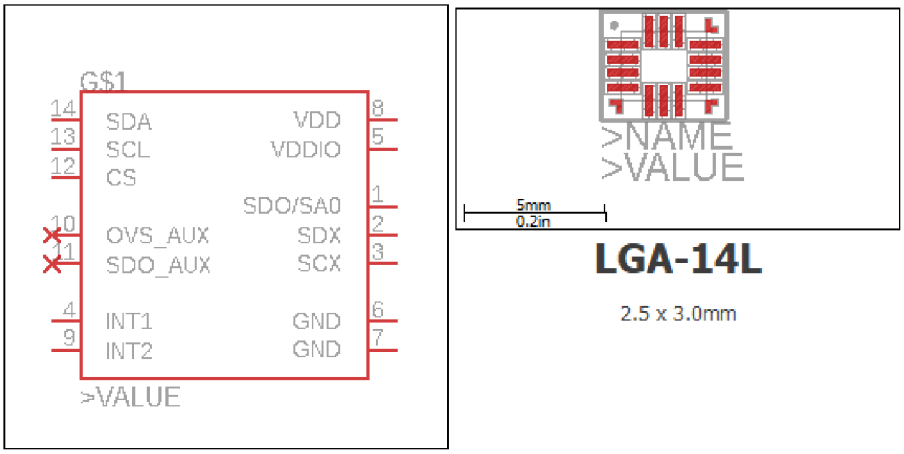
Aby bylo možné přidávat součástky do schématu, bylo potřeba ručně vytvořit většinu knihoven součástek. Knihovna Součástky (device) se skládá ze dvou částí – symbolu a footprintu.

Symbol určuje, jak součástka vypadá ve schématu. Určuje typ použitých pinů, jejich názvy a možnosti jejich prohození. Footprint představuje popis součástky na výsledné PCB. Obsahuje pady, díry, silkscreen potisk, určená místa pro nanášení pájecí pasty, lepidla, názvy, hodnoty a další informace. Spojením obou vzniká součástka (device), která umožňuje mapování na různé typy pouzder.

Knihovny se vytvářejí s použitím datasheetů, které obsahují typy jednotlivých pinů, rozměry pouzder a jejich doporučený footprint. Dle pokynů vedoucího byly u všech QFN i LGA pouzder nekrytou mědí ve tvaru křížků označeny rohy součástek (kromě rohu poblíž pinu č. 1), pro jejich správné osazení a kontrolu. Dále byly všechny pady prodlouženy o 0.5 mm za hranici pouzdra kvůli usnadnění procesu pájení a následné kontroly.

Některé knihovny byly dostupné na internetu. Osobní zkušenost však ukázala, že layout součástky ne vždy pasuje k datasheetu výrobce a bylo tedy nezbytné stažené knihovny kontrolovat proti datasheetu.

Téměř veškeré knihovny integrovaných obvodů, obsažených v této práci, byly vytvořeny v rámci práce zcela od začátku. Příklad knihovny s kontrolními křížky je k nahlédnutí na obrázku 3.4 – vlevo je symbol použitý ve schématu, vpravo pak fyzický layout s křížky.




Obrázek 3.4: Ukázka knihovny součástky (z programu Eagle CAD)

Dále byly použity knihovny pasivních součástek a pin headerů, dostupných v samotném nástroji Eagle.

### 3.3.3 Podpora pro debugování

Kromě zapracování SWD portu k debugování MCU, byly vyvedeny veškeré signály DWM1001 na pin headery pro jejich snadné připojení k osciloscopu, jejich měření a kontrolu. Mimo to obsahuje vývojový modul čtyři test-pady pro měření úrovně napájení.

### 3.3.4 Rozšíření

Vzhledem k nedostatkovému počtu pinů nebylo možné připojit veškeré výstupy použitých komponent k MCU. Z toho důvodu zůstalo záměrně nevyužito pět pinů MCU, z toho dva s možností analogových vstupů. Přebytečné (však nepotřebné) signály, které nebylo možné připojit (druhá přerušení IMU jednotek, výstupy nabíječe baterie, teplota baterie, přídavná tlačítka a přídavná LEDka) jsou poté vyvedeny v přídavném pin-headeru (J8). V případě jejich použití pak stačí propojit jejich výstupy s nevyužitými vstupy MCU. Díky těmto opatřením zůstala vývojová deska dále rozšiřitelná a funkcionalitu lze libovolně přidávat i bez přidávání GPIO expanderu. Pro snadné rozšíření o externí zařízení jsou pak na pin-header J7 vyvedené tři piny pro zem a tři piny pro jejich napájení.

### 3.3.5 Výsledné schéma

Výsledné schéma je pro lepší přehlednost logicky členěno na funkcionální části, jakými jsou periferie pro práci s uživatelem (LED, tlačítka, displej), $I^2C$ periferie, periferie zajišťující napájení (DC-DC, nabíječ) a konektory. Schéma je k nahlédnutí v příloze A.

Použité součástky se nachází v DPS kusovníku, který je k nahlédnutí v příloze B, v tabulkách B.1 a B.2.

## 3.4 Návrh motivu desky plošných spojů a její výroba

Po výběru součástek a návrhu schématu následuje výroba layoutu.

### 3.4.1 Rozmístění součástek a propojení

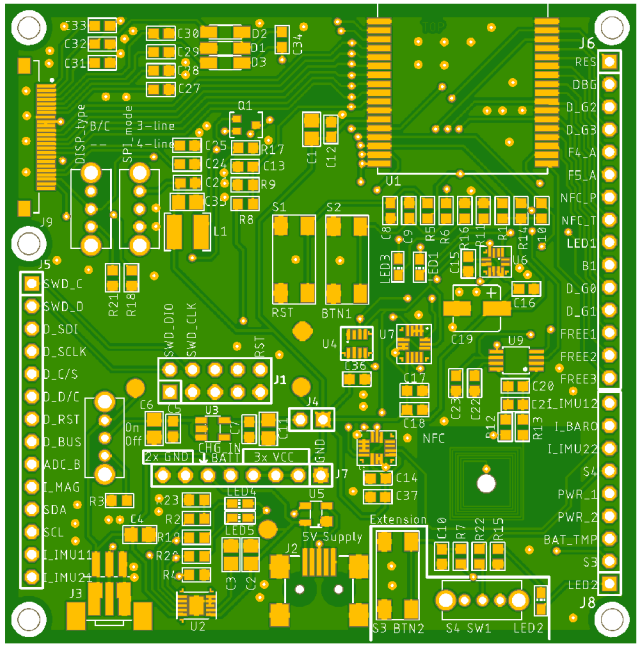
Součástky jsou rozmístěny primárně dle logické funkcionality, aby došlo k minimalizaci vzdálenosti v rámci daných skupin a minimalizaci rušení. Napájecí subsystém je umístěn tak, aby nezasahoval a nerušil ostatní subsystémy. Jsou zachovány dostatečné vzdálenosti mezi součástkami, aby bylo možné jejich ruční pájení. UWB anténu bylo potřeba dle datasheetu výrobce umístit tak, aby po obou stranách nedocházelo k rušení (z tohoto důvodu přesahuje okraj desky). Bylo minimalizováno použití různých průměrů vrtaných otvorů z důvodu snížení celkové ceny a urychlení výroby. Výsledné layouty vrchní i spodní vrstvy lze vidět na obrázcích 3.5 a 3.6.

### 3.4.2 Výrobní parametry

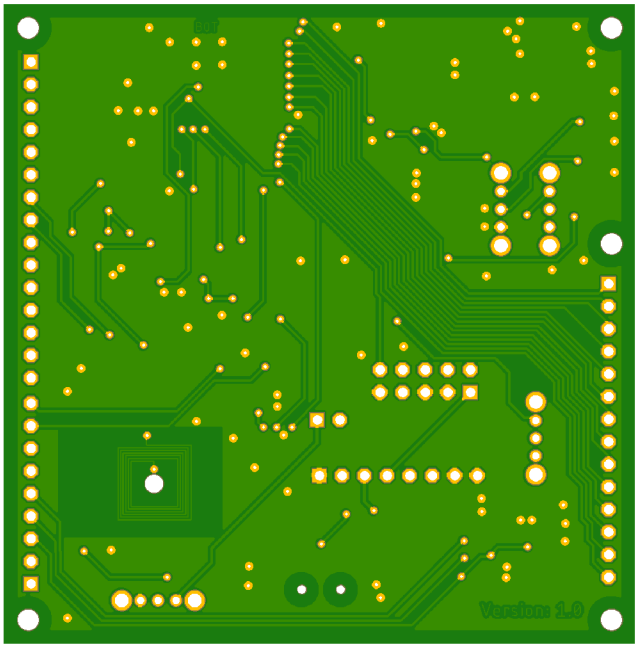
Kvůli zachování minimální ceny při výrobě byl zvolen dvouvrstvý layout. Jeho nevýhoda ovšem spočívá v nemožnosti rozlití napájení do samostatných vrstev a tedy minimalizování možných rušení.

Minimální šířka mezi vodiči, pady a prokovy je 0,2 mm. Minimální šířka vodičů je 0,2 mm. Minimální průměr vrtaných otvorů je taktéž 0,2 mm. V případě rozvodu napájení jsou všechny parametry nastaveny na 0,3 mm. Tloušťka desky je 1,5 mm. Tloušťka měděných rozvodů pak 18 µm. Z těchto parametrů lze dopočítat maximální proud, který může rozvody protékat. Pro obyčejné rozvody je maximální proud, při odchylce ± 10 °C a pokojové teplotě 25 °C, 210 mA. V případě napájecích rozvodů je hodnota 290 mA, což pro nabíjení i provoz ostatních částí postačuje.

Deska byla vyrobena společností Gatema s.r.o. K výrobě bylo třeba předat vygenerované soubory k layoutu v Gerber formátu a soubory s údaji pro vrtání.




Obrázek 3.5: Layout tagu, horní vrstva (z programu Eagle CAD)



Obrázek 3.6: Layout tagu, spodní vrstva (z programu Eagle CAD)

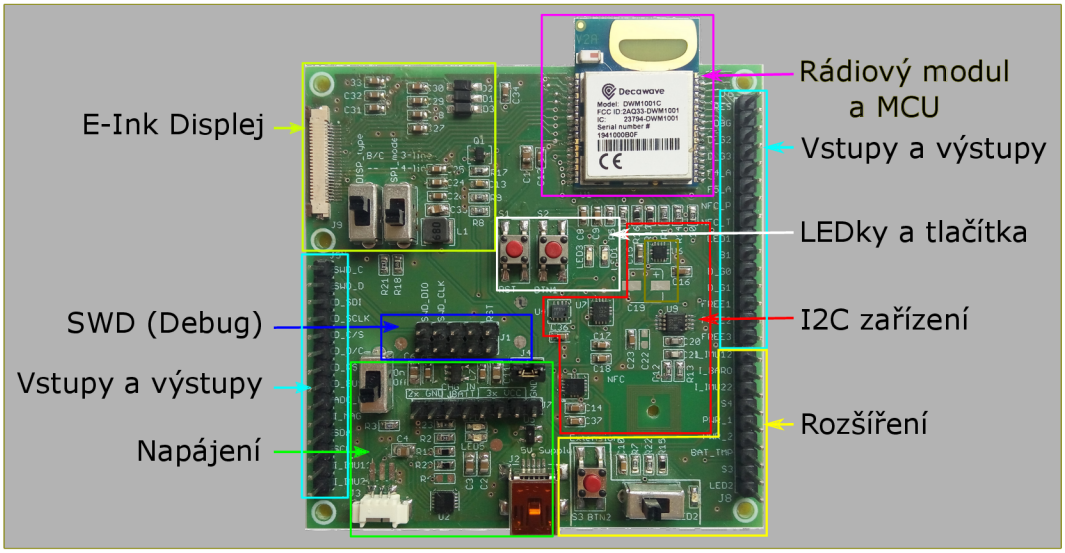
## 3.5 Osazení součástek a oživení modulu

Pájení desky proběhlo v laboratoři na FIT VUT v Brně za použití mikropáječky, horkovzdušné páječky a kompresoru pro dávkování tekutého cínu. Nejprve proběhlo pájení napájecího subsystému a jeho proměření. Poté došlo k pájení zbytku součástek. Nejproblematičtější bylo pájení součástek v malých LGA a QFN pouzdrech. Při jejich kontrole pomohly křížky odleptané mědi v jejich rozích.

Díky pečlivosti a několikanásobné kontrole při výrobě knihoven součástek, návrhu schématu, pokládání layoutu a pomoci vedoucího práce s pájením malých součástek se podařilo vyrobit funkční prototyp na první pokus.

K oživení desky byl vytvořen primitivní firmware, který bliká jednotlivými LEDkami. Problém nastal při nekompatibilitě školních debugerů s vývojovým studiem a procesorem. Z toho důvodu byl objednán a použit debugger J-Link EDU od společnosti SEGGER. K úvodnímu zjištění činnosti SWD na MCU a vymazání paměti byl použit nástroj J-Link Commander, verze 6.48.

Hotová deska, včetně popisů jednotlivých oblastí z kapitoly 3.4.1 je k dispozici na obrázku 3.7.



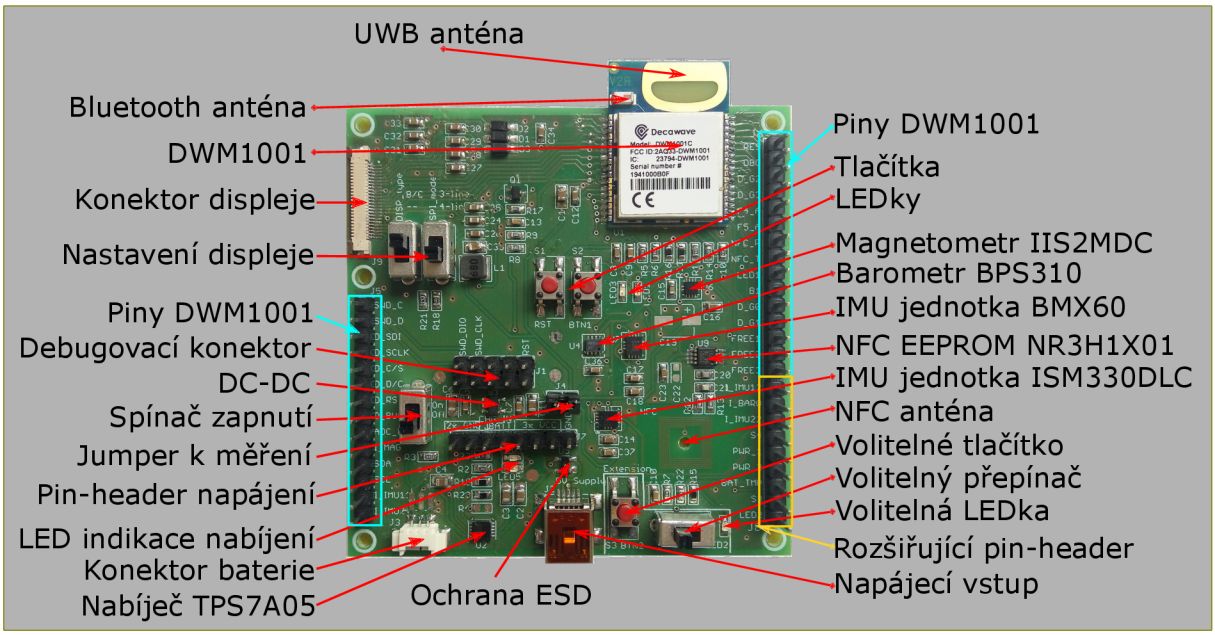

Obrázek 3.7: Výsledná vývojová deska s popisky oblastí

## 3.6 Výsledná podoba lokalizačního modulu

Finální podobu desky, včetně veškerých popisků jednotlivých součástek lze vidět na obrázku 3.8. Pro správné pochopení funkcionality desky je potřeba doplnit obrázek 3.8 dalším popisem:

- nastavení displeje – určuje zvolený mód SPI a zvolený typ displeje. Pro kompatibilní SPI s FW vývojového tagu je nutno nastavit levé tlačítko do horní polohy. Pro kompatibilitu se zvoleným displejem nastavit pravé tlačítko do polohy dolní;
- konektor displeje – konektor se připojuje odhalenými piny nahoru;

- tlačítka – levé tlačítko slouží k resetu MCU, pravé tlačítko slouží jako uživatelský vstup;
- rozšiřující pin-header – ke zprovoznění těchto pinů je nezbytné jejich propojení s neobsazenými piny MCU. Tyto piny jsou označeny předponou „FREE_“. Nutná je také úprava firmwaru;
- spínač zapnutí – tímto spínačem se zapíná/vypíná napájení celého tagu. Tento spínač neovlivňuje nabíjení – nabíjení baterie probíhá, i-když je spínač v poloze „vypnuto“;
- jumper k měření – slouží k připojení shunt rezistoru pro měření spotřeby proudu. Pro zapnutí desky musí zůstat sepnutý;
- pin-header napájení – lze použít k napájení externích periférií, volitelně připojených k vývojové desce.




Obrázek 3.8: Výsledná vývojová deska s popisky součástek

Pro detailní zapojení je k dispozici schéma v příloze A. Statistiky HW zapojení byly vygenerovány programem Eagle. Jsou zachyceny v tabulce 3.4.

| | |
|---|---|
| Rozměry desky | 71 × 72 mm |
| Celkem komponent (součástek) | 101 |
| Počet prokovů | 147 |
| Počet vrtů | 237 |
| Počet signálů | 84 |

Tabulka 3.4: Statistiky layoutu tagu

# Kapitola 4

# Návrh a implementace FW

Po návrhu HW tagu, jeho výrobě a popisu lze přistoupit k návrhu a implementaci FW[1] části. Ta se skládá ze specifikace požadavků, návrhu systému a samotné implementace. Během tohoto procesu vyvstalo několik překážek, které měly negativní dopad na průběh práce. Jejich popis a řešení jsou taktéž součástí textu.

## 4.1 Specifikace požadavků

Nejdůležitějším globálním požadavkem je nízká spotřeba. Dalšími požadavky jsou snadná rozšiřitelnost, přenositelnost a dobrá čitelnost kódu. V této podkapitole následuje podrobná specifikace jednotlivých funkcí vývojového tagu.

#### Odesílání UWB zpráv (blinků)

Hlavní funkcionalitou pro správné určení lokalizace tagu je odesílání blinků přes UWB. Formát těchto krátkých zpráv se řídí formátem zadavatelské společnosti z důvodu kompatibility a možnosti testování v jejich systému. Frekvence odesílání musí být nastavitelná, neboť pro různé aplikace se hodí různé intervaly (frekvence odesílání pro sledování hráče basketbalu při tréninku musí být ku příkladu vyšší, než frekvence odesílání pro sledování zboží ve skladu). Nezbytná je také možnost odesílání s náhodnou časovou odchylkou, aby se snížila pravděpodobnost kolize (překryvu, interference) zprávy se zprávou jiného tagu. Podporováno musí být několik typů zpráv, aby nedošlo k vytěžování rádiového média odesíláním přebytečných informací. V zadavatelské firmě pak existují následující typy zpráv.

**Info blink** slouží k ohlášení tagu v RTLS systému. Obsahuje veškeré informace o své konfiguraci a stavu baterie. Neobsahuje informace ze senzorů. Tento blink je odesílán třikrát ze startu tagu a poté každým 255. blinkem.

**Battery blink** slouží k odesílání údaje o stavu baterie. Podle zvoleného režimu k němu mohou být přibaleny i informace ze senzorů. Tato zpráva je poté odesílána každým patnáctým blinkem.

**Obyčejný blink** slouží primárně k určení pozice tagu. Může nicméně obsahovat také údaje ze senzorů. Je odeslán vždy, když nejsou odesílány ani info blink ani battery blink.

[1] Kdekoliv se v textu dále objeví termín aplikace, FW, nebo firmware, jedná se o synonyma k navrženému řídícímu programu popsanému v této sekci.

#### Měření baterie

Pokud je po delší dobu snížený stav nabití baterie, tag musí přejít do módu nízké spotřeby, ve kterém čeká, dokud se napětí na baterii nevrátí k normálu připojením nabíječky, nebo stabilizováním procesů v baterii. Taktéž je potřeba údaj o stavu pravidelně odesílat na RTLS server. Údaj může být posílán jako hodnota v milivoltech. Měření baterie by nemělo probíhat po jakékoliv akci, která má vysoký proudový odběr, aby nedošlo ke zkreslení údajů. Řešením může být potvrzování stavu na základě N po sobě jdoucích měření.

#### Sběr dat ze senzorů

Dále musí být možný průběžný sběr dat ze senzorů a jejich vyhodnocení. V závislosti na implementovaném chování musí být možné doby provádění těchto úkonů libovolně nastavovat.

#### Zobrazení provozních dat na E-Ink displeji

Displej by měl průběžně zobrazovat stav vývojového tagu. Stavem se rozumí dynamické informace získané za běhu a statické informace pro jeho snadnou identifikaci.

#### Indikace pomocí LEDek

K indikaci mimo displej slouží také LEDky. Jedna LEDka slouží k indikaci provozního stavu (LED3), vzhledem k požadavku nízké spotřeby by ale neměla dělat nic. K indikaci chyb nebo reakci na uživatelské vstupy by měla sloužit LED2.

#### Ovládání tagu

K ovládání tagu slouží vstupní tlačítka. Mělo by být možné různorodými sekvencemi zmáčknutí v libovolném stavu tagu provádět akce, které změní jeho chování.

#### Konfigurace tagu

Tag by mělo být možné konfigurovat. V zadavatelské společnosti je toho dosaženo pomocí UWB zpráv, které globální konfiguraci obsahují. Mělo by jít přidávat další kanály konfigurace.

#### Další rutiny

Aby bylo možné využít potenciálu vývojové desky, měl by FW umožňovat jednoduché přidání jakékoliv další rutiny (přidání dalších periferií a testování libovolné funkcionality).

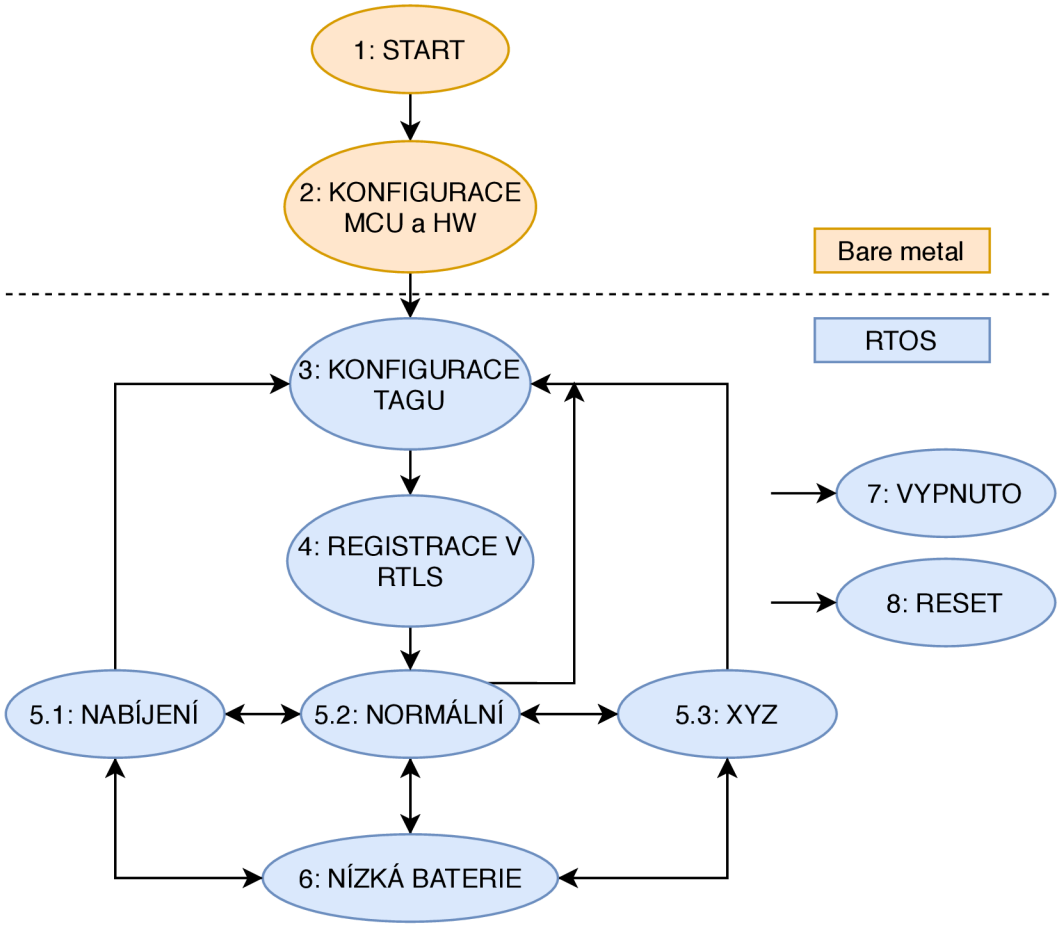
## 4.2 Návrh struktury firmware

Po specifikaci požadavků lze přistoupit k návrhu. Celý proces navrhování a implementace byl iterativní a prošel mnoha smyčkami. V této práci je dále popsán právě výsledek poslední iterace návrhu a implementace.

### 4.2.1 Stavový diagram

Celkové chování systému lze shrnout do následujícího stavového diagramu na obrázku 4.1. Ten je členěn do dvou částí:

- Bare metal (horní část) - slouží k provedení veškeré konfigurace před spuštěním operačního systému. Jedná se o nastavení periferií, knihoven, ovladačů, RTOS primitiv a inicializaci všech dalších struktur, potřebných k běhu aplikace.
- RTOS (spodní část) - slouží k implementaci samotného chování tagu, které je implementováno v RTOS systému.




Obrázek 4.1: Stavový diagram chování systému

#### Stav 1 – Start

Jedná se o provedení rutin ihned po spuštění. V případě Cortex M4 procesoru se z adresy 0x00 obnoví adresa ukazatele na zásobník. Poté dojde z adresy 0x04 k načtení obsluhy přerušení události reset a k jejímu vykonání. V tomto vykonávání dochází ke kopírování inicializovaných a neinicializovaných dat do RAM paměti MCU a provedení případných dalších konfigurací.

#### Stav 2 – Konfigurace MCU a HW

Jedná se o nastavení veškerých vnitřních periférií MCU (hodiny, přerušení, rozhraní, ...) i vnějších periférií vestavěného systému (E-ink displej, UWB modul, ...). Probíhá také nastavení kanálů pro výpis logovacích zpráv a inicializace knihoven.

#### Stav 3 – Konfigurace tagu

Jedná se o první stav v RTOS systému, který načítá konfiguraci přes příslušné médium a následně ji aplikuje. Jakmile dojde k aplikování této globální konfigurace, je potřeba restartovat činnost tagu a ohlásit aktuální nastavení RTLS systému znovu. Z toho důvodu za tímto stavem následuje stav registrace v RTLS.

Existuje několik způsobů, kterými lze konfiguraci získat:

- výchozí konfigurace – načtení konfigurace z FLASH paměti MCU;
- NFC – načtení konfigurace uložené přes NFC;
- UWB – příjem a načtení konfigurace přes UWB;
- Bluetooth – příjem a načtení konfigurace přes Bluetooth;
- EEPROM paměť – načtení poslední konfigurace z EEPROM paměti.

#### Stav 4 – Registrace v RTLS

Po každém spuštění tagu nebo přehrání novou konfigurací je nezbytné, aby se tag ohlásil RTLS systému zadavatelské společnosti. RTLS systém si uloží MAC adresu tagu a jeho aktuální platnou konfiguraci. K tomuto účelu slouží tři vyslání *info blink* zpráv, s rozestupem jedné sekundy. Ani tak ovšem není zajištěno doručení zprávy a úspěšná registrace tagu. Opakovaným odesíláním se však pravděpodobnost zvyšuje.

#### Stav 5.X – Mód běhu

Jakmile je tag nakonfigurován a ohlášen v RTLS systému, může fungovat v běžném módu. V tomto módu dochází k následujícím úlohám:

- odesílání UWB blinků (k určení polohy tagu a předání dat ze senzorů);
- měření údajů ze senzorů;
- měření aktuálního stavu baterie;
- výpis údajů na displej;
- běh dalších volitelných rutin.

V závislosti na daném podstavu (nabíjení, normální, ...) se poté upravuje četnost odesílání UWB zpráv, činnost senzorů a další chování.

#### Stav 6 – Nízká baterie

V tomto stavu tag přeruší jakoukoliv činnost a čeká na připojení nabíječky, vypnutí, nebo opětovný nárůst napětí na baterii.

#### Stav 7 – Vypnuto

Stav představuje vypnutí tagu, ve kterém je spotřeba minimální, a to díky vypnutí veškerých periferií. Opětovné zapnutí probíhá podržením tlačítka nebo připojením nabíječky. Do tohoto stavu lze přejít z jakéhokoliv jiného stavu.

#### Stav 8 – Reset

Tento stav slouží k resetování tagu, nejčastěji po detekci kritické chyby. Nejprve dochází k dokončení veškeré práce a následuje reset. Do tohoto stavu lze přejít z jakéhokoliv jiného stavu.

### 4.2.2 Aperiodické a periodické procesy

Pro další pokračování je potřeba specifikovat, jaké typy procesů systém obsahuje. Dle předchozí kapitoly 2.2.1 existují tasky, které se opakují (periodické), a tasky, které přicházejí náhodně (aperiodické). K navržení systému je zapotřebí u každého procesu také analyzovat, zda se jedná o soft task (nedodržení deadlinu reakce nezpůsobí škodu), nebo hard task (potřeba striktně dodržet deadline). Pokud procesy mají tyto deadliny, je nutno je definovat. Následuje rozřazení a specifikace procesů dle periodicity.

#### Periodické procesy

**Měření stavu baterie** je soft task, s variabilně definovanou periodou. Jediným omezením je nevhodnost měření před jakoukoliv UWB operací, která by mohla baterii na krátkou dobu vybít, a znepřesnit tak měření. U této operace není potřeba určovat deadline.

**Odesílání blinků** je hard task, frekvence odesílání blinků je dána uživatelem. Pokud by nedošlo k odeslání blinku ve stanovenou dobu, nebyla by zachycena poloha v RTLS systému (kupříkladu by se dělník s připevněným tagem blížil do nebezpečné zóny a systém by včas nestihl vypnout stroje, které by mohly vést k jeho zranění). Tento deadline je nastaven na dobu mezi dvěma následujícími blinky.

**Výpis údajů na displej** je soft task. Zpoždění zde není problém. Je také potřeba počítat s dlouhou dobou přepisu údajů na displeji, kvůli velikosti přenášených dat a e-ink technologii. Tento proces nemá stanovený deadline.

**Vyčítání údajů ze senzorů** je soft task. Jediným omezením je nutnost měření před samotným odesláním zprávy s naměřenými údaji. V případě snímání natočení tagu v prostoru se pak jedná o hard task, neboť aktuální natočení vždy závisí na předchozích měřeních (podobný princip jako u metody dead reckoning). Tato metoda ovšem není v této práci implementována.

#### Aperiodické procesy

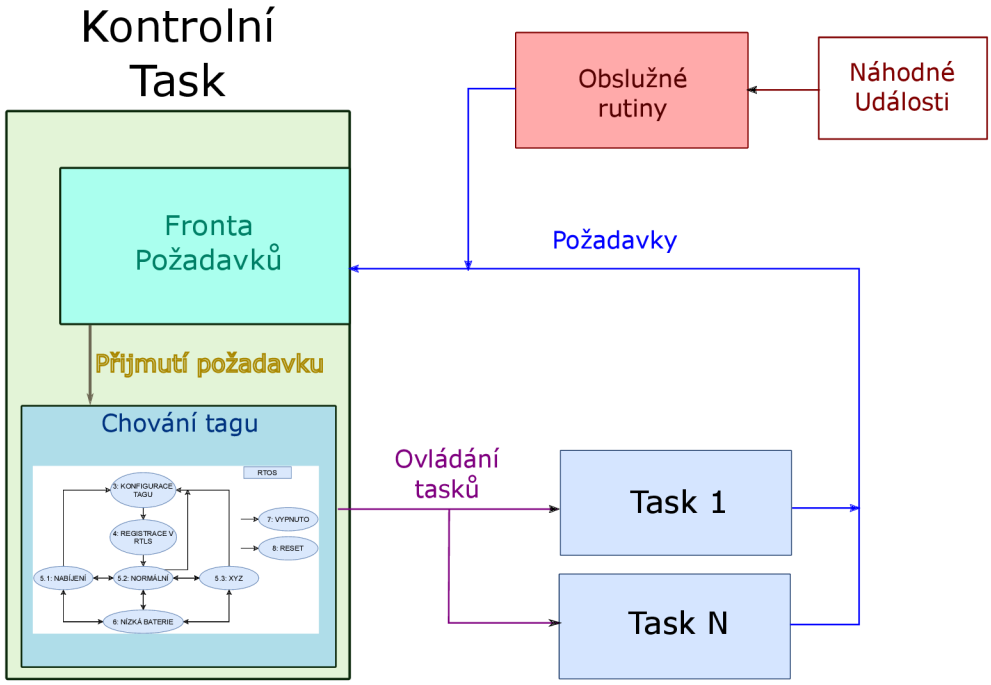
**Přerušení z tlačítka** je soft task. Operace stisknutí tlačítka budou dále upřesněny.

**Přerušení ze senzoru** je hard task. Dojde-li k detekci prahové hodnoty senzoru, je potřeba co nejrychleji adekvátně zareagovat.

**Detekce připojení nabíječky** je soft task, pomalé přepnutí do módu nabíjení nezpůsobí kritickou chybu.

### 4.2.3 Návrh chování s RTOS

Jak již bylo řečeno v teoretické části práce v sekci 2.2, RTOS je založen na přepínání tasků. Následující diagram 4.2 zachycuje návrh implementace chování tagu pomocí tasků.




Obrázek 4.2: Struktura chování programu tagu založeného na RTOS

Veškerá řízení zprostředkovává kontrolní task (vlevo). Ten funguje na principu konečného automatu, popsaného v podkapitole 4.2.1. Za běhu ovlivňuje chování dalších tasků (Task 1, . . . , Task N) dle aktuálního nastavení. Tento kontrolní task je ovlivněn vstupními událostmi, které mohou změnit interní chování tagu. Těmito událostmi jsou:

- hodnota nízkého stavu baterie, nebo stavu v normě;
- stisknutí implementované sekvence tlačítka;
- přerušení ze senzorů;
- připojení nabíječky;
- další volitelné události.

Z diagramu dále vyplývá, že veškerá implementace chování je definována právě v kontrolním tasku. Z této centralizace plyne, že lze jednoduše přidávat další funkcionalitu, a kód je lépe čitelný.

Fronta požadavků může být implementována jako obyčejná FIFO fronta. Je potřeba zajistit výlučný přístup při vkládání nových požadavků (ten je zajištěn automaticky funkcionalitou jádra FreeRTOSu). Kontrolní task poté ve smyčce čeká na další události, které by mohly změnit chování aplikace. Dle závažností událostí lze namísto přidání na konec fronty událost přidat na její začátek. Tím se zpracuje jako první.

## 4.3 Programové vybavení

K implementaci byly zvoleny jazyky C (C11) a Assembler. K překladu byly použity nástroje GNU ARM, verze 5.4 (červen, 2016). Následuje výčet použitých nástrojů, knihoven a technologií. Jejich použitím vznikla celá řada komplikací, kterým bylo potřeba během vývoje čelit. Komplikace i řešení jsou taktéž popsány v následujících sekcích.

### 4.3.1 Vývojové prostředí

Dle doporučení Nordic Semiconductor, výrobce použitého čipu MCU, bylo možné zdarma využít nástroj Segger Embedded Studio, verze 4.50. Na základě doporučení byla polovina práce psána s použitím tohoto nástroje. Při používání byly objeveny mnohé bugy, kvůli kterým se nástroj stal dále nepoužitelným:

- Nejvýraznějším bugem byla navigace v projektu, kde mnohdy nebylo možné využít funkce „jít na definici“, nebo „jít na deklaraci“. Tento bug nešel nijak obejít ani opravit.
- Dalším bugem byla nesprávná interpretace funkce „jít do souboru“, například při přepnutí na hlavičkový soubor. Funkce zobrazila obsah souboru, který se však nenacházel na aktuálním kurzoru.
- Třetím nepříjemným bugem byly nefunkční výpisy debugovacího okna, které si neumělo poradit s některými sekvencemi pro nový řádek (LF) a návrat na první znak (CR). Díky tomuto bugu mizely celé zprávy. Až po velice důkladném debugování uvnitř zdrojových kódů k SEGGER RTT bylo zjištěno, že zakáže-li se znak CR na určitých místech, IDE interpretuje a vypíše debugovací zprávu správně.

V polovině vývoje bylo kvůli bugům nezbytně nutné přejít na IDE Embitz, ve verzi 1.11 [33]. Toto IDE je open-source a nepodporuje použitý procesor NRF52832. Bylo tedy nezbytné manuálně nastavit veškeré informace a dodat soubor „nrf52.svd“.

### 4.3.2 Možnosti debugování

Vzhledem k nekompatibilitě školních debugerů s vývojovým studiem bylo potřeba objednat J-Link EDU od společnosti SEGGER. Podporuje jak SWD, tak J-link technologie. Jeho následné využití a využití laboratoře pak zkomplikovala opatření ohledně pandemie COVID-19.

#### SEGGER RTT

K výpisu debugovacích zpráv byla použita technologie SEGGER RTT [34] (Real Time Transfer). Tato technologie funguje podobně jako semihosting – odesílá údaje přes J-link/SWD rozhraní. Oproti semihostingu však poskytuje výhodu, že minimálně narušuje běh programu. Pomocí SWD/J-link je průběžně v době debugování vyčítán blok SEGGER RTT Control Block v RAM paměti MCU. V této paměti se nachází ID (k automatické identifikaci adresy bloku při zahájení debugování) a deskriptory bufferů pro komunikaci oběma směry (vstup, výstup). V rámci SEGGER RTT lze mít otevřeno zároveň několik kanálů. V projektu jsou pak využity kanály dva. Kanál pro SEGGER RTT (debugovací zprávy) a kanál pro SEGGER SystemView (debugovací zprávy FreeRTOS). Ve výchozí konfiguraci je staticky vyhrazeno místo v paměti pro kanál 0.

Existují tři módy běhu SEGGER RTT, které lze konfigurovat pro každý kanál: vypnuto, kruhové přepisování a čekání na přečtení. Tyto módy definují, co se má stát v případě, že je buffer plný (ztráta dat přepisováním nebo čekání na vyprázdnění bufferu).

Při použití této technologie je potřeba nastavit správné velikosti bufferů. Velikosti bufferů ovlivňují celkové množství dostupné paměti.

Veškeré konfigurace se pak nacházejí v souboru *SEGGER_RTT_Conf.h*.

#### SEGGER SystemView

Tento nástroj slouží k zaznamenávání událostí a analýze běžícího vestavěného systému. Je postaven na technologii SEGGER RTT. Voláním inicializační funkce si dynamicky alokuje kanál 1. Stejně jako SEGGER RTT jej lze nakonfigurovat na daný mód běhu a měnit velikosti bufferů. Oproti SEGGER RTT odesílá zprávy v přesně stanoveném formátu, což umožňuje jejich rozdílné interpretace tímto nástrojem, a zobrazení užitečných informací.

V tomto projektu je nástroj použit pro analýzu chování FreeRTOS systému. Nástroj umožňuje zobrazit časovou osu, události s přesným časem jejich příchodu (spuštění plánovače, přerušení časovače, ...), a jejich analýzu, viz kapitola 5.

Pro jeho použití bylo nezbytné provést úpravu FreeRTOS souborů (patchování). Těmito úpravami se rozumí úpravy volání rutin, které vyšlou zprávu přes SEGGER RTT v případě, že nastala některá systémová volání. Těmito voláními jsou například přechody tasků do stavu připraven, spuštění plánovače, přerušení časovače, úprava systémových struktur, a další. Vzhledem k nekompatibilitě použitého portu FreeRTOS operačního systému bylo nutné provést patchování v rámci práce manuálně, a to procházením jednotlivých souborů.

Veškeré konfigurace jsou dostupné v souboru *SEGGER_SYSVIEW_Conf*.h

### 4.3.3 Knihovna NRF SDK (sfotware deployment kit)

Celý vývoj by nebyl možný bez podpory výrobce MCU. Ten připravil řadu knihoven a příkladů, které bylo možné při vývoji použít. Pro vývoj byla vybrána nejnovější verze NRF SDK 15.3.0 [35], podporující daný procesor a jeho revizi. Nevýhodou této verze bylo, že obsahovala kompletně novou sadu a způsob použití knihoven pro NRF52, ovšem s neustálou podporou staré verze. Toto SDK tedy obsahovalo verze dvě, což ze začátku komplikovalo jejich použití ve vývoji.

Nejdůležitějším souborem celého SDK je *sdk_config.h*. Tento soubor obsahuje velké množství nastavení (soubor o délce 12 000 řádků). Myšlenkou je centrální nastavení pro všechny knihovny i drivery daného SDK. Pro nastavení/povolení/zakázání prakticky čehokoliv je tedy potřeba upravit nastavení tohoto souboru.

### 4.3.4 Použitá distrubice FreeRTOS

Byl použit FreeRTOS ve verzi 10.0.0, který byl obsažen v rámci SDK výrobce. Tato kapitola velice zběžně popisuje jeho funkce, nezbytné k pochopení finální implementace.

**Systémové hodiny RTOSu** řídí chod celého operačního systému (přepínání kontextu, časový údaj o aktuálním taktu). Pokud by četnost spouštění těchto hodin byla příliš vysoká, docházelo by k vysoké režii (především při preemptivním přepínání a funkcí time-slicing). Pokud by ale četnost byla příliš malá, mohlo by docházet k nízké odezvě. Bez těchto hodin by FreeRTOS nemohl fungovat. Port poskytnutý výrobcem nabízí možnost výběru mezi dvěma systémovými časovači: SysTick (součástí Cortex M4

architektury), nebo RTC1 časovač (součást NRF52832, jako periferie). Dle datasheetu výrobce procesoru je použití RTC1 časovače z pohledu nízké spotřeby vhodnější.

**Tickless mód** umožňuje vypnout systémové hodiny RTOSu a přepnout procesor do režimu spánku po dobu, kdy není naplánována žádná událost.

**Plánovač** FreeRTOSu umožňuje prioritní preemptivní plánování s funkcí time-slicing. K time-slicingu dochází v případě, kdy jsou připraveny k běhu dva procesy se stejnou prioritou. Time-slicing střídavě přiděluje procesor oběma procesům. Tuto funkci lze zakázat.

**Detekce přetečení zásobníku** je další funkce FreeRTOSu. Probíhá nastavením posledních položek zásobníku na specifické hodnoty. Dojde-li k přepisu těchto hodnot, znamená to, že došlo k přetečení zásobníku. Detekce probíhá při přepínání kontextu.

**SW timer** slouží k periodickému, nebo jednorázovému spouštění rutin po uběhnutí časového kvanta. Jeho vykonávání probíhá pomocí dalšího běžícího tasku *timer task*, který je spuštěný při startu FreeRTOSu. Tento task obsahuje frontu požadavků, která časovače ovládá (spuštění časovače, přidání, odebrání, modifikace). Nevýhodou tohoto přístupu je, že se příkaz provede až po přepnutí kontextu na tento task. Výhodou je, že je celý časovač vystavěn na již existujících primitivech FreeRTOSu.

**IDLE task** se spouští při startu FreeRTOSu. Slouží jako pojistka, kdyby v systému nebyl žádný běžící task. Zároveň se s jeho spuštěním uvolňuje paměť. Není-li nic na práci, dojde ke spuštění tohoto tasku. Na jeho principu je poté vystavěna funkce tickless módu.

**Prioritu** má přiřazen každý proces FreeRTOSu. Čím vyšší číslo, tím vyšší priorita. Na základě této priority se poté rozhoduje, který proces má být přepnut do běžícího stavu.

Při konfiguraci je nutno správně nastavit priority přerušení. V praxi se může stát, že přijde přerušení MCU. To získá procesor a chce volat funkce FreeRTOSu. K takové situaci nesmí dojít, neboť se může stát, že aktuálně běžící task je právě uprostřed vykonávání systémového kódu. Kdyby v tento moment nastalo přerušení s voláním systémového kódu, došlo by k nekonzistenci systému. Z toho důvodu se musí před kterýmkoliv voláním systémového kódu maskovat všechna přerušení, která mohou volat systémové funkce.

**Semafory, fronty a mutexy** jsou další podporovaná primitiva. Mutexy navíc umožňují řešení problému priority inversion pomocí algoritmu priority inheritance.

### Nastavení FreeRTOSu

V rámci tohoto projektu bylo zvoleno následující nastavení (dostupné v souboru FreeRTOSConfig.h):

- zvolení systémového časovače RTC1 s frekvencí 1000 Hz;
- preemptivní plánování s vypnutím time-slicingu;
- povolení tickless módu, ...

Zbytek nastavení lze dohledat v přiloženém souboru *FreeRTOSConfig.h*.

### 4.3.5 Systémová knihovna CMSIS

Aby byl kód alespoň částečně přenositelný, byla použita vrstva CMSIS Core [36] (Cortex Microcontroller Software Interface Standard, Core). Tato vrstva poskytuje jednotná rozhraní k ovládání NVIC (správa přerušení), SysTicku (ARM Cortex M časovač), jednotné pojmenování registrů a instrukcí, systémových výjimek v rámci různých výrobců MCU, založených na ARM Cortex M architektuře. Napomáhá tak ke snazší přenositelnosti kódu. Vzhledem k tomu, že výrobce nepodporuje další CMSIS vrstvy a manuální přepisování ovladačů by bylo značně časově náročné, je CMSIS Core jedinou použitou vrstvou.

## 4.4 Struktura projektu

Následuje popis struktury projektu. Ta zajišťuje jeho snadnou čitelnost, rozšiřitelnost a přenositelnost. Díky vysvětlení těchto základních prvků bude čtení dalších kapitol a procházení souborů projektu jasnější.

### 4.4.1 Názvy souborů

Aby bylo možné pouze z názvu hlavičkového souboru určit, jaké má závislosti, zda je přenositelný a do jaké patří kategorie (co přibližně obsahuje), byly zavedeny následující předpony:

**drv_** (driver) – zařazuje soubor do kategorie ovladačů. Tyto soubory by neměly mít jakoukoliv návaznost na aplikační kód a použitou platformu. Jedná se o zcela přenositelné jednotky. Svou přenositelnost zajišťují externími funkcemi, které je potřeba pro danou aplikaci přibalit. Veškeré takovéto implementace se nachází ve složce drivers/bindings.

**lib_** (library) – zařazuje soubor do kategorie knihoven. Tyto soubory by měly být taktéž nezávislé na aplikačním kódu a použité platformě. Implementují cokoliv, co není závislé na aktuálním HW a pracuje čistě na softwarové úrovni (například knihovna pro generování CRC součtů).

**com_** (common) – zařazuje soubor do kategorie společných souborů pro různé projekty. Obsahují údaje (struktury, konstanty, definice, funkce, ...), které jsou společné pro různé typy projektů. Neobsahují žádné závislosti.

**app_** (application specific) – zařazuje soubor do kategorie aplikačně specifické. Takovéto soubory jsou závislé na aktuální platformě a konkrétní funkcionalitě. Představují implementaci chování aplikace. Jejich přenositelnost závisí na způsobu jejich implementace.

**cnf_** (configuration) – zařazuje soubor do kategorie konfigurací. Obsahuje konfigurační položky, které mění chování aplikace.

Z tohoto výčtu je patrné, že přenositelné jsou následující kategorie: ovladače (drv_), knihovny (lib_) a kategorie společných (com_). Aplikačně závislé jsou naopak kategorie: aplikačně závislých (app_) a konfigurace (cnf_).

### 4.4.2 Souborová struktura

Projekt je rozdělen do následující souborové struktury:

- *app_specific* – obsahuje aplikačně specifický kód;

- *common* – obsahuje kód společný pro více projektů;
- *config* – obsahuje konfiguraci aplikace;
- *drivers* – obsahuje ovladače;
- *libraries* – obsahuje knihovny;
- *linker* – obsahuje soubory pro linkování (definice layoutu paměti);
- *startup* – obsahuje soubory s kódem, které se spouští před spuštěním funkce `main()` a vektory přerušení;
- *third_parties* – obsahuje knihovny třetích stran (freeRTOS, RTT);
- *../nRF_SDK_15.3.0* – obsahuje SDK pro daný mikrokontrolér.

### 4.4.3 Názvy konstrukcí jazyka

Vzhledem k tomu, že je projekt psán v jazycích C a Assembler, neexistuje žádná možnost použití vlastnosti jmenných prostorů (namespace) z jazyka C++. K přiblížení této funkcionality slouží názvy souborů a dalších konstrukcí jazyka. Je potřeba zajistit, aby veškeré globální konstrukty byly v rámci celého projektu unikátní. K zajištění unikátnosti slouží přidání předpony. Předpona musí jednoznačně identifikovat daný celek a zároveň musí korespondovat s názvem souboru, aby byl kód lépe čitelný.

Příkladem názvu souboru je drv_bmx160.h (jedná se o IMU jednotku, její označení je použito v názvu). Funkcí je pak kupříkladu `bmx160_wakeUp()`.

## 4.5 Implementace jádra aplikace

Dle návrhu popsaném v kapitole 4.2 byla vytvořena implementace systému. V následujících podkapitolách je implementace popsána detailněji. Pro připomenutí je k dispozici struktura systému na obrázku 4.3, zachycující strukturu chování aplikace.

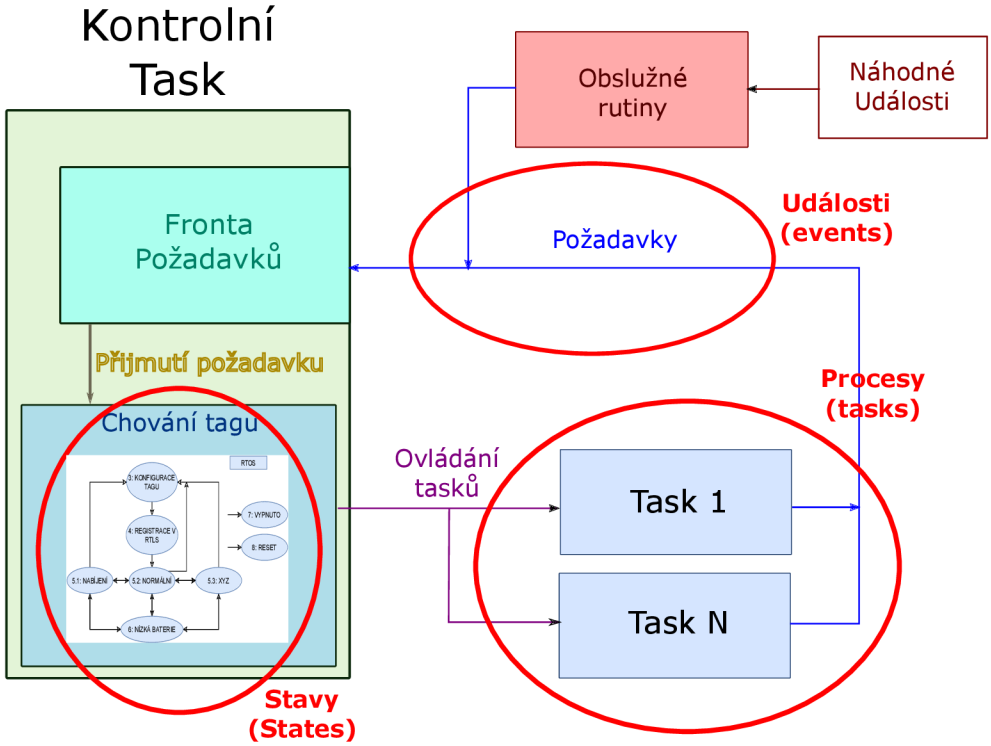
### 4.5.1 Řešení paralelně běžících tasků

Jak již bylo popsáno dříve, systém ovládá kontrolní task. Ten přijímá požadavky z čela fronty a na základě nich a stavu, ve kterém se nachází, pak provádí dané akce. Akcí se také rozumí řízení dalších, paralelně běžících, tasků. V systému dále pracuje *idle task*, který uspává systém, nemá-li nic na práci, a *timer task*, který obstarává funkcionalitu softwarového časovače.

Pomocí tohoto *timer tasku* jsou také implementovány pseudo-paralelně běžící procesy. *Timer task* umožňuje běh několika časovačů zároveň a to jak periodických, tak jednorázových. Umožňuje také za běhu měnit jejich stav a upravovat dobu spuštění. Při expiraci daného časovače dochází ke spuštění handleru (funkce) daného časovače a při současném běhu více časovačů je tím zajištěn běh všech potřebných částí.

### 4.5.2 Sdílená data

Aplikace obsahuje mnoho sdílených dat a jejich předávání mezi funkcemi vytváří hůře rozšiřitelný a čitelný kód. Předávání dat do funkcí je také pomalejší (kopírování dat na zásobník) a není možné přistoupit k datům z funkce přerušení.




Obrázek 4.3: Struktura chování programu s RTOS

Z toho důvodu byla zvolena globální struktura `tag_runtime_t`, která obsahuje další podstruktury, které člení sdílená data do kategorií. Pro každou kategorii existují data konfigurační (použitý UWB kanál, ...) a data, která se mění za běhu (teplota, napětí, ...). Struktura obsahuje následující kategorie:

- `tag_sensors_t` – data ze senzorů;
- `tag_peri_t` – data dalších periférií;
- `tag_uwb_t` – data k UWB komunikaci;
- `tag_tdoa_t` – data k TDOA algoritmu;
- `tag_system_t` – systémová data;
- `tag_peri_t` – freeRTOS data;
- `tag_others_t` – data nespadající do žádné z kategorií.

Díky rozdělení do kategorií je také možné implementovat výlučný přístup pro přístup do každé z nich.

Veškerým globálním proměnným v programu byla přidělena předpona *g_* (od slova global). Struktura těchto sdílených dat dostupná v programu se poté nazývá `g_tag` a je dostupná v souboru *app_tag_runtime.h*. Pro práci s ní je nezbytné zahrnout tento hlavičkový soubor do zdrojového textu.

### 4.5.3 Implementace jádra

Veškerou implementaci jádra aplikace lze nalézt v souboru *main.c* a zdrojových souborech ve složce *app_specific/tag_behavior*. Jak je patrno z obrázku 4.3, lze implementaci jádra rozdělit na tasky, události a stavy. Každé z těchto primitiv je implementováno v dané složce, ve zdrojových souborech s názvy *app_behav_events* (události), *app_behav_states* (stavy) a *app_behav_tasks* (tasky). Soubor *main.c* poté obsahuje základní stavový automat chování aplikace.

#### Události (events)

Události mohou změnit chování systému. Zasílají se kontrolnímu tasku, který provede odpovídající reakci. Jedná se například o zmáčknutí dané sekvence tlačítka, nízkou hodnotu baterie a další. Události (představuje je struktura `behav_event_t`) se skládají ze dvou částí: kategorie a podkategorie. Rozděleny jsou tak z důvodu snížení režie při procházení stavového prostoru při jejich zpracování.

Hlavní kategorie, dostupné jako primitivum enum s názvem `behav_events_global_t` jsou následující:

- `BEHAV_EVENT_BATTERY` (události baterie – napájení);
- `BEHAV_EVENT_BUTTON` (události tlačítka);
- `BEHAV_EVENT_PERIPHERAL` (události z periférií);
- `BEHAV_EVENT_CONFIG` (události konfigurace);
- `BEHAV_EVENT_STATE` (události ostatní).

Každá z těchto kategorií obsahuje podkategorie. Příkladem může být kategorie tlačítka `behav_events_button_t`:

- `BEHAV_EVENT_BUTTON_ONOFF` – sekvence zmáčknutí k zapnutí a vypnutí tagu;
- `BEHAV_EVENT_BUTTON_ACTUALIZE` – sekvence zmáčknutí k aktualizaci stavu displeje.

Hlavičkový soubor *app_behav_events.h* obsahuje jednotlivé definice událostí a zdrojový soubor *app_behav_events.c* pak reakce na ně. V případě rozšíření stavového prostoru stačí následovat schéma v obou souborech.

Zdrojový soubor dále poskytuje řídící funkce:

- `behav_event_getEvent(...)` – blokující funkce, implementující čtení příchozí události z fronty požadavků;
- `behav_event_sendEvent(...)` – odeslání nově příchozí události do fronty požadavků;
- `behav_event_isEvent(...)` – zjednodušující funkce, zajišťující snadnější porovnávání.

#### Stavy (states)

Hlavní *kontrolní task* se může nacházet v jednom ze stavů, popsaných dříve v podkapitole 4.2.1. Přepínání do jednotlivých stavů ovlivňují příchozí události a přirozený chod programu.

Stavy jsou popsány pomocí primitiva enum s názvem `behav_states_t`. Implementace každého stavu je rozdělena do tří funkcí:

- začátek (start) – kód proveden před vstupem do stavu (inicializace stavu);
- tělo (body) – kód prováděn periodicky;
- konec (end) – kód proveden před změnou stavu (de-inicializace stavu).

Startem stavu se rozumí nastavení jednotlivých parametrů běhu pseudo-paralelně prováděných úkonů (Task1, ... TaskN). Jedná se například o nastavení frekvence blinků a jejich opakované odesílání. V těle tasku poté kontrolní task čeká na příchozí události a provádí příslušné akce. Reakce a čekání se provádí v nekonečné smyčce, dokud není změněn stav programu. Po změně dojde ke spuštění konečné (end) funkce stavu a k přepnutí na jiný stav, který opět začíná počáteční (start) funkcí.

Příkladem může být stav naíjení (charging). Je implementován pomocí funkcí: `behav_state_charging_start` (start), `behav_state_charging_body` (tělo), `behav_state_charging_finalize` (konec).

Veškeré tyto funkce a definice stavů lze nalézt v souborech *app_behav_states.c* a *app_behav_states.h*.

Tyto soubory dále poskytují řídící funkce:

- `behav_state_changeState(...)` – funkce, která mění globální stav tagu;
- `behav_state_getState(...)` – funkce, která vrací aktuální stav tagu;
- `behav_state_getStateName(...)` – funkce, vracející název aktuálního stavu pro jeho možný výpis.

#### Tasky (tasks)

Obsahuje implementace pseudo-paralelně běžících tasků. Dle konkrétní implementace jsou tyto tasky definovány pomocí opakovaných či jednorázových expiračních funkcí časovače. Implementaci lze však snadno přetvořit na implementaci pomocí skutečných tasků, dle potřeb konkrétní aplikace. Nutné je poté upravit start těchto tasků ve všech stavech a jejich inicializaci před startem FreeRTOS plánovače.

Implementace tasků lze nalézt v souborech *app_behav_tasks.c* a *app_behav_tasks.h*

## 4.6 Implementace aplikace

K jádru byla postupně vystavena celá aplikace. Jedná se o skupinu ovladačů, knihoven a promyšlených struktur takových, aby byl kód dále rozšiřitelný, přenositelný a snadno čitelný. V této podkapitole jsou kromě nich dále popsány techniky pro dosažení nízké spotřeby a problémy a zajímavosti, které vyvstaly při implementaci.

### 4.6.1 Ovladače HW periférií

Periférie v tomto projektu jsou fyzické integrované obvody, které, stejně jako jejich zdrojové kódy, mohou být použity v dalším projektu. Za tímto účelem bylo nutné kromě fyzické implementace ovladačů provést také implementaci aplikačně specifických externích funkcí a definici pinů. Tato implementace umožňuje jejich použití pro danou platformu. Implementace pro každou periférii se poté nachází ve složce *drivers/bindings*. Pro každou platformu je nezbytné implementaci doplnit/pozměnit pomocí maker s doplněním platformě specifického kódu.

### E-Ink displej

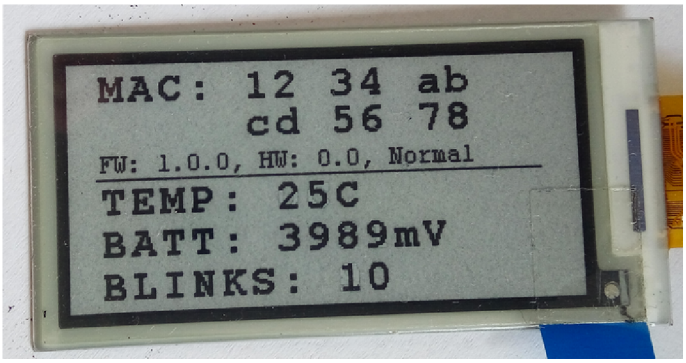
Zajímavostí displeje výrobce Waveshare [32] byla podpora standardního rozhraní SPI. Při bližším prozkoumání datasheetu však bylo zjištěno, že toto rozhraní nemá s SPI téměř nic společného. SPI rozhraní, jak jej výrobce Waveshare nesprávně nazývá, nelze přímo ovládat SPI rozhraním MCU (rozhraní definováno společností Motorola). Toto rozhraní displeje neobsahuje *MISO* pin a pomocí *MOSI* pinu odesílá a zároveň přijímá data. S rozhraním od Motoroly sdílí pin *CS*, který je synonymem pro pin *Slave Select*. Pokud je pin v log. 0, je cílové zařízení aktivováno k přenosu dat. Dále sdílí vodič *SCL*, který udává synchronní vzorkování přenášených bajtů (dle zvoleného režimu 8, nebo 9 bitů).

Ovládání displeje se provádí prostřednictvím odesílání příkazů (COMMAND) a dat (DATA). Nejprve je odesláním příkazu vybrán registr, do kterého má být hodnota zapsána, a následují data, která hodnotu změní/nastaví. SPI rozhraní obsahuje dva módy. Ty určují, jakým způsobem displeji dáváme vědět, zda přenášený Bajt znázorňuje příkaz, nebo data.

První mód využívá speciální vodič *D/C*. Pokud je signál v log. 0, jsou přenášená data interpretována jako příkaz. Pokud je signál v log. 1, jedná se o data. Druhý mód tento vodič nevyužívá. K přenášenému bajtu musí být připojen na začátek dodatečný bit, který říká, zda se jedná o data, nebo příkaz.

Díky této neshodě bylo potřeba naimplementovat vlastní ovladač, který emuluje výrobcovo SPI pomocí GPIO mikrokontroléru.

Implementace chování E-ink displeje je následovná: ze startu programu se provede jeho vymazání (tři probliknutí) a následné nastavení na parciální překreslování. Dle nastavené periody se na něj průběžně zobrazují: napětí baterie, pořadí aktuálního blinku, teplota, MAC adresa tagu k jeho snadné identifikaci, verze aktuálního HW a FW a aktuální stav (mód) tagu. Ukázky displeje lze nalézt na obrázcích 4.4 a 4.5 .

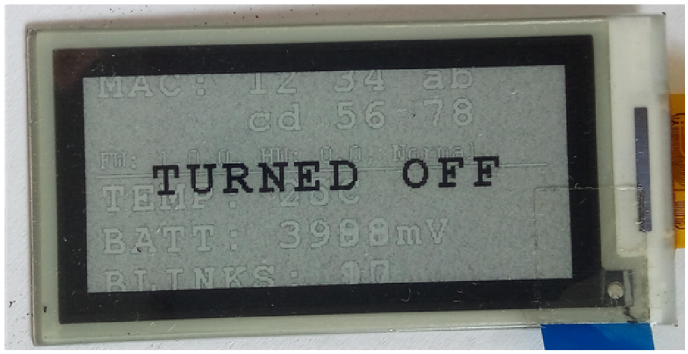



Obrázek 4.4: Ukázka obrazovky displeje s vykreslenými údaji v běžných stavech

Byla využita knihovna výrobce [32], která umožňuje základní zobrazení textu a tvarů. Její implementace však není ideální. Nejasný kód a magické konstanty znesnadňovaly její použití a především debugování.

### NFC EEPROM

Jedná se o EEPROM paměť, která může být přepsána/vyčítána pomocí technologie NFC, nebo přes $I^2C$ sběrnici. Díky použití NFC technologie může dojít k přepsání paměti, aniž by byl čip fyzicky napájen, neboť je napájen z bezdrátové komunikace. Přepisovat pomocí NFC lze většinu paměťového rozsahu, který lze taktéž zamykat proti neoprávněnému přepisu.




Obrázek 4.5: Ukázka obrazovky displeje ve stavu vypnuto

Zajímavostí čipu je komunikační protokol. Po vystavení adresy čipu na $I^2C$ sběrnici je očekáváno potvrzení pomocí operace ACK na datovém vodiči SDA. Toto potvrzení provedly veškeré ostatní čipy na desce (po jejich adresaci) a byla tak detekovatelná jejich přítomnost. Tento čip však vyžadoval také zápis adresy paměti, na niž probíhá zápis, nebo čtení dat. Až po zápisu adresy docházelo k potvrzení dat pomocí operace ACK.

NFC EEPROM není aktuálně v projektu používána. K paměti byl v rámci práce napsán ovladač dle datasheetu výrobce a je tak možné ji kdykoliv použít.

#### DW1000

Čip pro UWB komunikaci je nejsložitější periferií z celé navrhované desky. Bylo možné získat knihovnu pro jeho ovládání ze stránek výrobce [37]. Pro její zprovoznění bylo potřeba doimplementovat startovací sekvenci, sekvenci pro probuzení ze spánku a přenos dat pomocí SPI. Nastavení UWB parametrů čipu bylo provedeno po vzoru zadavatelské společnosti z důvodu zachování kompatibility s jejich systémem.

### 4.6.2 Ovladače periférií a knihoven

Tato sekce obsahuje velice stručný popis dalších ovladačů a knihoven, implementovaných a použitých v rámci tohoto projektu.

#### Knihovna pro LEDky

Tato knihovna slouží k ovládání LED diod. Statická konfigurace zajišťuje snížení režie v době běhu. Zajímavostí je, že umožňuje vyblikat jakýkoliv vzor. Vzory jsou pak definovány jako posloupnost příkazů zapnout/vypnout s dobou čekání mezi nimi. Vzory je poté možné libovolně opakovat. Funguje na bázi FreeRTOS časovače, který implementuje pauzy mezi danými příkazy. Umožňuje taktéž blokující, nebo neblokující mód.

#### Knihovna pro tlačítka

Tato aplikační knihovna slouží k extrakci vzorů zmáčknutí tlačítka. Různé vzory poté v aplikaci představují různé typy událostí. Podporuje dva vzory režimů: režim držení tlačítka, který funguje za pomoci časového intervalu, ve kterém musí být tlačítko drženo a režim počtu stisků tlačítka, kdy je nejen definován časový interval, ale také je zjišťován počet stisků v tomto režimu.

V tomto projektu jsou implementovány dva typy stisknutí. První představuje aktualizaci dat na displeji při podržení tlačítka v intervalu 100–2000 ms. Druhé představuje zapnutí a vypnutí tagu při podržení tlačítka v intervalu 2000–N ms.

#### Další knihovny

Byly implementovány i další knihovny: CRC součty, náhodný generátor, knihovna pro přerušení, knihovna pro teplotní čidlo, aplikační knihovna pro ovládání displeje a EEPROM paměti.

Pro ovládání mikrokontroléru byly použity knihovny výrobce, dostupné v NRF SDK 15.3.0 [35].

### 4.6.3 Konfigurace za běhu

Vývojový tag lze konfigurovat několika způsoby: pomocí bluetooth, NFC EEPROM paměti a UWB. Aktuální implementace umožňuje konfiguraci z EEPROM paměti mikrokontroléru. Další konfigurace lze snadno přidat následováním schématu konfigurací v *app_specific/config/*. Pro ovládání NFC EEPROM paměti lze využít naimplementovanou knihovnu pro daný čip a aplikační knihovnu pro přepisování konkrétních bloků paměti, která obsahuje předpřipravené paměťové schéma, které lze použít. Schéma se skládá z výchozí konfigurace, aktuální konfigurace a konfigurace, kterou lze nahrát přes NFC paměť. Výchozí konfiguraci lze změnit úpravou aktuálně použité konfigurace v souboru *config/tag/cnf_global_config.h*.

### 4.6.4 Statická konfigurace

Implementace statické konfigurace se nachází ve složce *config/tag*. Hlavní soubor konfigurace chování tagu je *cnf_global_all.h*. Tento soubor postupně definuje konfigurace podkategorií, vkládáním dalších konfiguračních souborů:

- *cnf_global_config.h* – umožňuje měnit aktuální funkcionalitu tagu;
- *cnf_global_debug.h* – umožňuje měnit nastavení logovacích zpráv;
- *cnf_global_pins.h* – umožňuje měnit piny jednotlivých periferií.

Samotný globální soubor konfigurace *cnf_global_all.h* dále obsahuje definici verzí HW, FW a platformy. Myšlenkou tohoto způsobu konfigurací je přidávání maker a konfigurací dle aktuálně zvolené platformy.

Dalšími konfiguracemi jsou konfigurace SDK (popsána dříve), konfigurace RTOSu a RTT. Poslední dvě konfigurace obsahují konfigurační soubory, které jsou přidány do virtuální složky projektu */config*. Fyzicky se však nacházejí ve složkách svých knihoven. Díky virtuálním složkám, implementovaných ve vývojovém studiu Embitz, lze mít v projektu přehledně veškeré konfigurace v jedné virtuální složce.

### 4.6.5 Přenositelnost

Kromě implementace předchozích opatření prostřednictvím názvů, předpon, souborové struktury a aplikačně specifického kódu ovladačů, které byly popsány dříve, bylo pro lepší přenositelnost vytvořeno schéma portů.

Aby bylo možné přenést aplikačně specifický kód i na jiná zařízení a rapidně tak snížit dobu vývoje, je nezbytné vytvořit abstraktní vrstvu mezi použitým HW a kódem. Takovým rozhraním může být například CMSIS. Výrobce použitého procesoru však podporuje pouze CMSIS core.

Proto byly vytvořeny porty, nacházející se ve složce *common/ports*. Základem je globální soubor *com_port_global.h*, který je možné vložit do každého zdrojového kódu prostřednictvím direktivy include a jednotně využívat jeho funkce (ovládání pinů, uspávání, reset, logovací funkce). Tento soubor představuje rozhraní, které musí veškeré porty pro jednotlivé platformy implementovat. Výběr konkrétního portu je proveden pomocí globálního makra, nastaveného v definicích při překladu. Volání funkcí jsou implementována formou maker, veškerá režie tedy probíhá za překladu.

### 4.6.6 Logování a debugování

Byla implementována podpora logování na třech hlavních úrovních:

- SDK výrobce – umožňuje logování pomocí technologií UART a RTT;
- FreeRTOS systému – implementováno pomocí Segger SystemView, používající technologii RTT;
- uživatelských výpisů – používající technologii RTT.

Logování lze zapnout v implementovaném souboru *cnf_global_debug.h*. V něm lze povolit jednotlivé úrovně nastavováním maker: `LOGGING_FREERTOS_ENABLED`, `LOGGING_SDK_ENABLED` a `LOGGING_USER_ENABLED`. Ke globálnímu zakázání všech logování slouží makro `LOGGING_ALL_ENABLED`.

Uživatelská logování byla implementována formou modulů. Každý modul, který má být logován, musí obsahovat vložení hlavičkového souboru *com_all.h*, který vkládá globální funkce portu. Před tento soubor je nezbytné vložit makro se jménem, pod jakým má být soubor logován, a pomocí něhož bude možné jeho logování povolovat nebo zakazovat.

Příkladem může být soubor *main.c*. Makrem pro definici jména budiž `#define DEBUG_MODULE_NAME MAIN`. Díky této akci mají veškeré uživatelské logy z tohoto souboru předponu „MAIN_“ a lze je zakázat nebo povolit v konfiguraci debugování pomocí `DEBUG_MAIN_LOGGING_ENABLED`, skládající se z předpony „DEBUG_“ a přípony „_LOGGING_ENABLED“.

Funkce pro logování zpráv se pak nachází v globálním portu, jako `DEBUG_PRINTF(...)`. Tento mechanismus je plně přenositelný díky implementaci v globálním portu. V případě nedefinování jména dostane zpráva předponu „UNDEF_“, kterou je možné měnit.

Po zapnutí uživatelských logování lze pomocí nástroje J-Link RTT Viewer přijímat zprávy povolených modulů, které informují o aktuálním stavu systému.

### 4.6.7 Nízká spotřeba

Nízké spotřeby bylo ve FW dosaženo za použití následujících technik:

- uspávání externích zařízení při jejich nečinnosti;
- použití tickless režimu ve FreeRTOSu;
- použití nízko-frekvenčních hodin procesoru (LFCKL);

- automatické řízení spotřeby procesoru při nečinnosti jeho periférií;
- použití low-power režimu přerušení GPIO pinů mikrokontroléru;
- využití časovače RTC1 procesoru NRF52862 namísto časovače SysTick.

Použití low-power režimu přerušení GPIO pinů mikrokontrolrué nemohlo být úspěšně provedeno, neboť došlo k jeho zničení při pokusech. Tento režim funguje na základě aktivity LDETECT signálu, který sdružuje veškeré změny provedené na sledovaných vstupech. Při zjišťování, který pin aktivitu provedl, pak používá softwarový průchod porovnávání předchozího a aktuálního stavu pinů. Opačným módem k tomuto GPIO režimu je možnost detekce přerušení vstupů ve standardním režimu. V tomto režimu však umožňuje detekci pouze osmi pinů, namísto očekávané podpory všech pinů. Tento režim je namísto low-power režimu při testování dále použit, což ovšem zvyšuje reálnou spotřebu tagu.

### 4.6.8 Priority

Priority tasků byly přiřazeny následovně: nejvyšší prioritu má *timer task*, který musí být schopen plnit systém případnými událostmi a dodržovat časové požadavky implementace. Současně zajišťuje výlučný přístup mezi pseudo-paralelně běžícími procesy, neboť v něm může běžet právě jeden z nich. Nejnižší prioritu má *idle task*, který pracuje v době, kdy nemá systém nic jiného na práci. Střední prioritu má kontrolní task, který provádí řízení systému a reaguje na vstupní události.

### 4.6.9 Možnosti přenositelnosti a rozšíření

Projekt je možné dále rozšiřovat za pomoci dříve popsaných, implemnetovaných technik. Následuje rekapitulace a zásady, které je potřeba při přenesení, nebo jeho rozšíření dodržovat:

1. Při přenositelnosti aplikace na jinou architekturu je nezbytné implementovat nový port v souboru *com_port_global.h*. Příkladem nového portu může být port *com_port_CPU123.h*, vyplněném po vzoru portu *com_port_nrf52832.h*. Porty jednotně zpřístupňují funkce debugování, ovládání GPIO, čekání a resetu. Ideálně by pak měly implementovat veškeré pro architekturu specifické funkce pro ovládání periferií, po vzoru CMSIS.

2. Do nového projektu lze importovat cokoliv ze složky *common/*, která představuje sdílenou, neměnnou funkcionalitu, přístupnou pro všechny projekty. Neměla by obsahovat žádný kód, který by byl určen pouze pro specifickou platformu. Totéž platí o složce *libraries/*.

3. V novém projektu lze taktéž použít ovladače ze složky *drivers/*. Tyto ovladače ovšem potřebují určení pinů, rozhraní a funkce pro jejich ovládání. Toho lze docílit jejich implementací ve složce *drivers/bindings* a s použití sdílených definic pinů v souboru *cnf_global_pins.h*.

4. Pro každou novou platformu je potřeba přidat její typ do globální definice, vytvořit její samostatný build v nastavení projektu a implementovat pomocí maker její funkcionalitu v jednotlivých souborech.

### 4.6.10 Podpora více vývojových desek

Během implementace této práce bylo potřeba vyvíjet Firmware i v době, kdy HW tagu ještě nebyl vyroben. K tomu sloužila deska DWM1001 Development Board [38], znázorněna na obrázku 4.6. Firmware byl navržen s ohledem na přidávání dalších budoucích platforem. Podpora dané desky je řešena pomocí definice globálního makra a postupů uvedených dříve. Aktuálně jsou podporované dvě desky: deska, vyvinutá v rámci této práce a DWM1001 development board. V programu je představují globální makra *BOARD_TAG_THESIS* a *BOARD_DWM1001_DB*. Pro vygenerování správného firmware je potřeba přepnout správnou build konfiguraci ve vývojovém studiu.


Obrázek 4.6: DWM1001 Development Board, převzato z [38]

# Kapitola 5

# Testování a spotřeba

Po úspěšném vytvoření firmware tagu následuje jeho testování. Testování lze rozdělit na dvě kategorie: testování proudové spotřeby a testování stability firmware.

## 5.1 Stabilita firmware

Testování stability firmware není triviální úloha. Tento pojem může představovat řadu významů. Stabilitou firmware v této práci se rozumí:

- běh v maximálním (ideálně nekonečném) čase, se zachováním definovaných funkcionalit aplikace;
- schopnost reagovat na neočekávané situace se zachováním definovaných funkcionalit.

Funkcionalitami firmware se v real-time systému myslí především dodržení deadlinů. Taktéž pak provádění veškerých úloh podle specifikace. Ve firmware byly implementovány tři mechanismy, detekující nestabilitu firmware:

- detekce přetečení zásobníků jednotlivých tasků (stack overflow);
- detekce nedostatku paměti při dynamické alokaci malloc;
- kontrola návratových hodnot funkcí a adekvátní reakce (kontrola knihoven, ovladačů).

Veškeré tyto chybové stavy lze detekovat za pomoci logovacích výpisů a byly hojně využity při vývoji. Jakmile k nim dojde, dochází k resetování firmware.

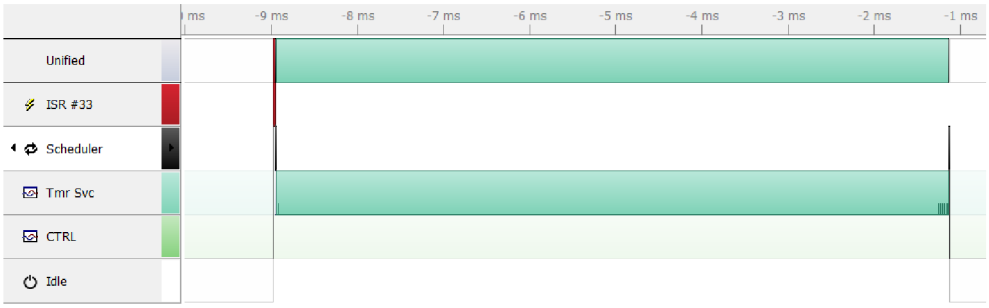
Další možnou komplikací jsou race conditions. Ty mohou změnit funkcionalitu programu tak, že dojde k porušení deadlinů nesprávným pořadím vykonávání, nebo současným přístupem ke zdrojům, na které platí výlučný přístup. Varianta výlučného přístupu může být kontrolována pomocí návratových hodnot funkcí při současné kontrole chybových nekonzistentních hodnot.

K detekování první varianty porušení deadlinů lze přistoupit různými způsoby. Jedním z řešení může být kontrola pomocí GPIO. Každá příchozí událost a běžící task by měl dedikovaný GPIO výstup. Ten by byl nastavován podle toho, kdy událost přišla a zda task aktuálně využívá CPU. Tyto výstupy by byly snímány logickým analyzátorem. Externím programem by bylo kontrolováno správné dodržení deadlinů a pořadí. Další variantou je využití RTT kanálu, který ovšem přidává do programu další režii spojenou s kopírováním zpráv do paměti.

V této práci je jediným taskem s pevně daným deadlinem pouze odesílání blinků. Kontrola dodržení deadlinu byla provedena za pomoci aplikace SEGGER SystemView.

### Odesílání blinku

Provedení blinku trvá přibližně 8 ms. Do této doby není započítáno probuzení a uspávání MCU. Tato doba obsahuje měření ze senzorů, přípravu dat blinku, výpočet další doby odeslání blinku spolu s náhodnou odchylkou, nastavení časovače na další spuštění odesílání blinku a jeho odeslání do DW1000 přes SPI s rychlostí 4 MHz. Přesná ukázka je vidět na obrázku 5.1, kde zelená výplň představuje proces odesílání blinku.




Obrázek 5.1: Proces odesílání blinku (z aplikace Segger SystemView)

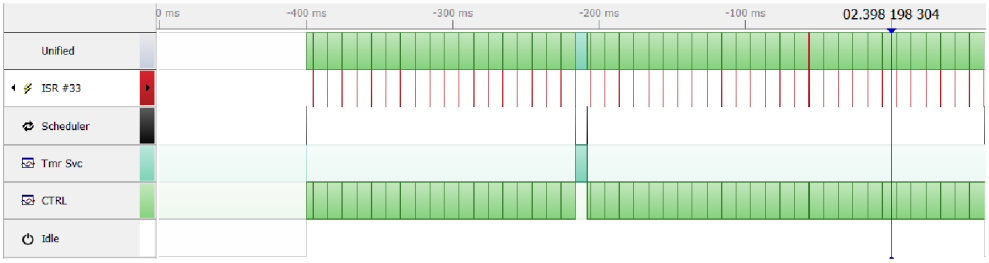
Z tohoto výsledku vyplývá, že není možné mít periodu odesílání bliku nižší, než 8 ms. Toto je v pořádku, neboť minimální perioda blinků je 10 ms.

### Výpis údajů na displej

Provedení výpisu údajů na displej trvá přibližně 450 ms (z toho 120 ms trvá příprava dat a 330 ms samotné odesílání). Tento časový okamžik představuje přípravu displejových dat a odeslání údajů na displej pomocí emulovaného SPI rozhraní. Přesná ukázka je vidět na obrázku 5.2, kde zelená výplň představuje proces vypisování dat na displej.


Obrázek 5.2: Proces vypisování dat na displej (z aplikace Segger SystemView)

Díky tomu, že proces odesílání blinků je nadřazen výpisům displeje, dojde k přepnutí kontextu i v průběhu odesílání displejových dat, je-li to potřeba, jak je patrno z obrázku 5.3 (provádění odesílání displejových dat je znázorněno zelenou barvou, odesílání blinku tmavě zelenou).




Obrázek 5.3: Proces vypisování dat na displej (z aplikace Segger SystemView)

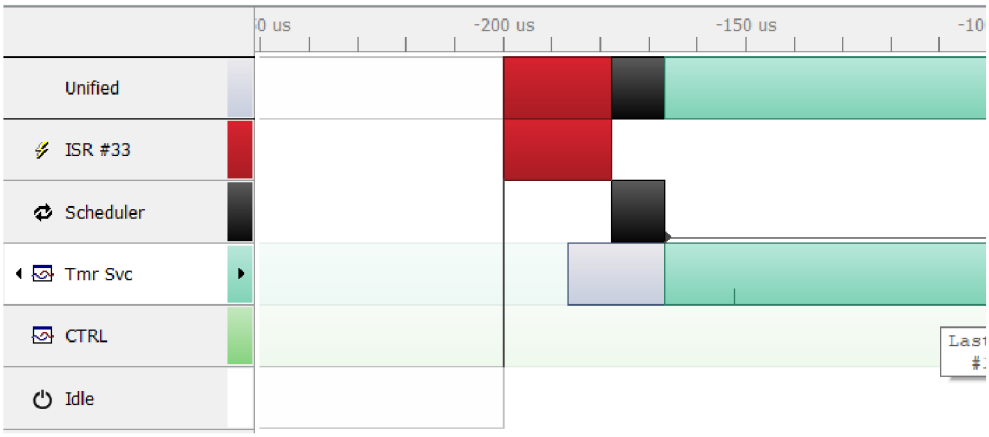
Odesílání dat na displej je provedeno za pomoci emulovaného nestandardního SPI rozhraní výrobce s maximální rychlostí blížící se 20 MHz. Této rychlosti ovšem nebylo možno dosáhnout. Pro snížení režie firmwaru při vykonávání odesílání by mohly být použity následující periferie MCU: PPI, standardní SPI s HW úpravami, časovače a EasyDMA.

#### Další události

Předešlé události jsou z časového pohledu nejnáročnějšími v systému. V systému se dále nachází událost zpracování vstupů a přepínání úloh.

Přepínání úloh po probuzení MCU trvá přibližně 35 µs. Tato událost by neměla narušit časový deadline blinků. Přesná ukázka je vidět na obrázku 5.4, kde černá výplň představuje přepínání kontextu a červená běh přerušení, kdy probíhá detekce změny kontextu.

Délka reakce na vstupní události GPIO (zmáčknutí tlačítka) je v průměru 200 µs.




Obrázek 5.4: Proces přepínání úloh (z aplikace Segger SystemView)

### 5.1.1 Vyhodnocení stability firmware

Bylo implementováno několik metod pro měření stability Firmware, které umožnily kontrolu správnosti aplikace. Pomocí aplikace Segger SystemView byla provedena časová analýza běžících úloh. Z té vyplynulo, že časové požadavky při odesílání blinků by neměly být ničím zarušeny při stávající konfiguraci a jednoduché funkcionalitě.

Bylo zjištěno, že nejvíc vytěžuje MCU výpis údajů na displej. Výpis trvá 450 ms. Závisí pak na konkrétní aplikaci, zda to bude problém (při sledování palety zboží se displej aktualizuje v době přiřazení tagu k paletě. Může trvat dny, než může být aktualizován znovu). Dopad by se nicméně dal zmírnit použitím HW periferií MCU, namísto SW emulace komunikačního protokolu. Dalším řešením je pozastavení komunikace a provedením jiných tasků tak, aby nedošlo ke ztrátě kontextu komunikace.

Doba odeslání blinku trvá přibližně 8 ms. Tuto dobu lze taktéž zkrátit použitím DMA mikrokontroléru.

## 5.2 Proudová spotřeba

Z předchozích kapitol je zřejmé, že spotřeba firmware se odvíjí od toho, v jakém stavu je řídící/kontrolní task. Pokud je tag na nabíječce (stav nabíjení), dochází k odesílání blinků s nižší frekvencí, než když je firmware ve stavu normální. Pokud je firmware vypnutý (stav vypnutý), měla by být spotřeba minimální.

Firmware lze dále dekomponovat. Ve všech RTOS stavech je zapnutý běh operačního systému. V případě, že dojde k vstupním událostem, spotřeba vzroste.

Když se nebavíme o vypnutém stavu ani o stavu kritického množství baterie, opakují se následující úlohy:

- odesílání blinků s danou frekvencí – nutnost probudit DW1000 a provést odeslání UWB zprávy;
- kontrola stavu baterie – nutnost provést měření pomocí ADC;
- měření údajů ze senzorů – dle typu senzoru se spotřeba různí;
- výpis údajů na displej – při parciálním módu se na spotřebě dosti podílí.

FreeRTOS může zásadním způsobem ovlivnit celkovou spotřebu následovně:

- Tickless mód – v případě jeho zapnutí nedochází k pravidelným přerušením časovače při nečinosti, jedná se tedy o nejzásadnější účinnou metodu ke snížení spotřeby v této aplikaci.
- Frekvence přepínání – k pravidelným přerušením časovače dochází v průběhu aplikace, aby včas došlo ke správnému přepínání tasků podle jejich priorit a zachování odezvy systému. Pokud je frekvence vysoká, může to vést k lepší odezvě, pokud nízká, dochází k menší režii a tedy spotřebě.
- Práce s frontou – v implementaci se nacházejí dvě fronty: fronta pro časovač a fronta pro ovlivnění funkcionality kontrolního tasku. Efektivita použití tohoto primitiva má také vliv na celkovou spotřebu.
- Práce s časovačem – v systému se nachází velké množství časovačů. Časovač je interně implementován formou tasku. Nepřímo tedy závisí na rychlosti přepínání tasků a odesílání a příjmu událostí z fronty, pomocí které je ovládán. Taktéž pak algoritmem řazení časovačů.

### Módy procesoru NRF52832

Mikrokontrolér NRF52832 disponuje jednotkou *Power management unit*. Tato jednotka automaticky vypíná a zapíná napájení a hodiny periférií tak, aby došlo k minimální spotřebě.

Po resetu se systém nachází v *SYSTEM ON* módu. Tento mód nabízí dva nízko-příkonové módy: nízko-příkonový (low power) a konstantně-latenční (constant latency). První mód zajišťuje nižší spotřebu, ovšem negarantuje stejnou latenci při probuzení MCU. Druhý mód garantuje konstantní latenci, ale má vyšší spotřebu.

Nejnižší spotřebu má systém v *SYSTEM OFF* módu. Z tohoto módu je možné systém probudit pouze pomocí GPIO, komparátoru, nebo NFC modulu. Nevýhodou je, že po probuzení následuje reset systému.

### Naměřené hodnoty

Z datasheetů výrobců čipů lze stanovit, které čipy způsobují nejvyšší spotřebu:

- MCU v IDLE módu spotřebovává v průměru 4-8 mA, v SYSTEM ON nízko-příkonovém módu spotřebovává v průměru 1.5 µA.
- DW1000 ve sleep módu spotřebovává 1 µA, v IDLE módu 19 mA a při vysílání 70 mA.
- E-ink displej by měl během operace spotřebovávat 4.5 mA. Ve sleep módu pak 2 µA.

Z těchto hodnot je patrná špičková spotřeba při vysílání UWB zpráv a současném vykreslování údajů na displej.

Za pomoci ampérmetru, s průměrováním výsledků po 0.5 s, bylo provedeno měření proudové spotřeby v místě pro shunt rezistor na HW desky. Veškeré logovací výpisy byly vypnuty, blikání LEDky při odesílání zprávy bylo zakázáno, a debugger po naprogramování danou verzí pokaždé vypnut. Displej zůstal nepoužit.

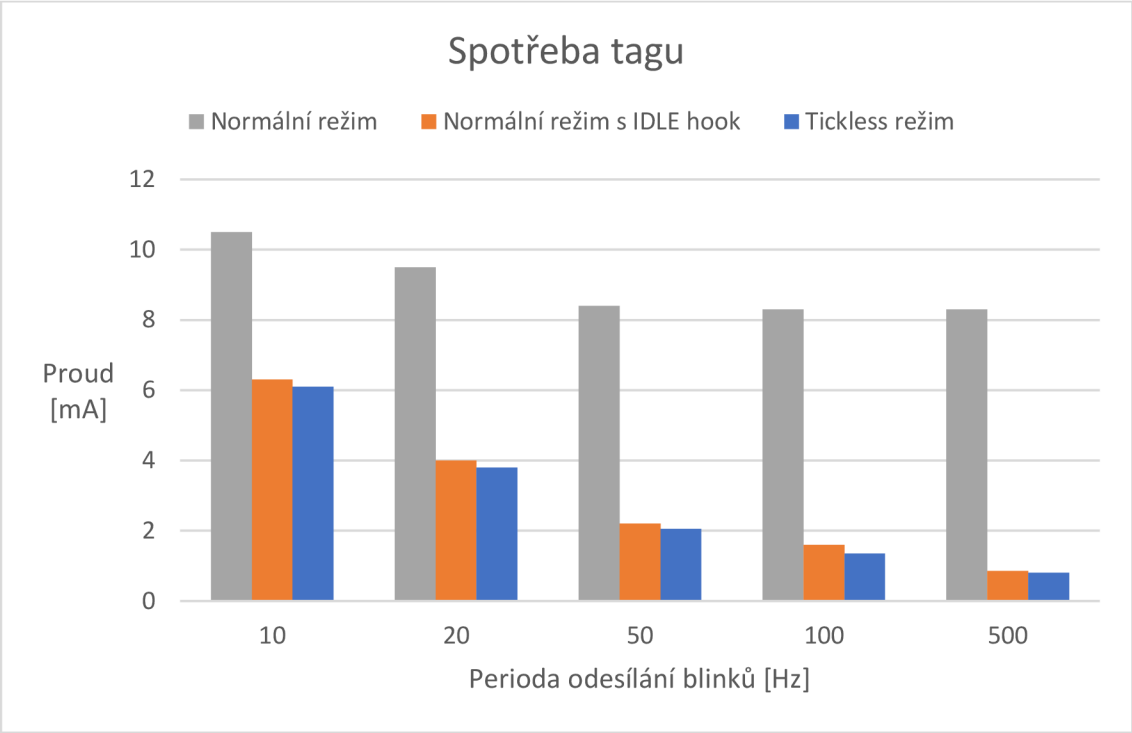
Periody odesílání blinku byly stanoveny na 10 ms, 20 ms, 50 ms, 100 ms, 500 ms. Zároveň s tím byl režim FreeRTOSu použit ve třech módech:

- Tickless režim – režim FreeRTOS s vypnutím časovače v době spánku.
- Normální režim s IDLE hook – normální režim s vložením instrukce spánku do tasku IDLE.
- Normální režim – režim bez přechodu do režimu spánku.

Všechny režimy byly nastaveny s frekvencí RTC1 časovače FeeRTOSu 1000 Hz. Výsledek měření je znázorněn v tabulce 5.1 a grafu 5.5.

| | **10 ms** | **20 ms** | **50 ms** | **100 ms** | **500 ms** |
|---|---|---|---|---|---|
| Tickless režim | 6,1 mA | 3,8 mA | 2,05 mA | 1,35 mA | 0,8 mA |
| Normální režim s IDLE hook | 6,3 mA | 4,0 mA | 2,2 mA | 1,6 mA | 0,85 mA |
| Normální režim | 10,5 mA | 9,5 mA | 8,4 mA | 8,3 mA | 8,3 mA |

Tabulka 5.1: Naměřená spotřeba




Obrázek 5.5: Graf naměřené spotřeby

### 5.2.1 Vyhodnocení proudové spotřeby

Z naměřených hodnot je patrné, že v tickless režimu se snižující se frekvencí odesílání UWB zpráv celková spotřeba rapidně klesá. Úplně stejné je to v módu normálním s IDLE hook režimem. Je to dáno tím, že v obou módech dochází k přepnutí do režimu spánku. Při bližším prozkoumání je patrné, že Tickless režim je vždy úspornější než jeho konkurent. Je to dáno tím, že v úsporném režimu tickless módu neprobíhá neustálé vzorkování časovače.

Naproti tomu spotřeba se snižující se frekvencí u normálního režimu klesá velice málo. Je to dáno tím, že operační systém nepřechází do stavu nízké spotřeby, ale nadále probíhá vzorkování časovače 1 kHz bez přechodu do režimu spánku.

Při vypnutí veškerých podporovaných periferií (režim vypnutí tagu) dosahuje spotřeba tagu pouhých 580 µA. Dle zjištěných hodnot spotřeby proudu z datasheetu výrobců by měla být spotřeba daleko nižší. Tato nepřesnost je dána dalšími přítomnými periferiemi na desce, které odebírají proud bez přepnutí do úsporných módů. Jedná se o periferie, které byly přidány na desku jako součást rozšíření práce.

Na samotný závěr následuje tabulka 5.2, která zobrazuje maximální dobu používání tagu v nepřetržitém provozu s použitou baterií o kapacitě 300 mAh a nejúspornějším režimem.

| **Perioda blinků** | **10 ms** | **20 ms** | **50 ms** | **100 ms** | **500 ms** |
|---|---|---|---|---|---|
| Výdrž (dny, zaokrouhleně) | 2 dny | 3 dny | 6 dní | 9 dní | 16 dní |

Tabulka 5.2: Doba provozu dle zvolené frekvence odesílání blinku (mód s nejnižší spotřebou)

# Kapitola 6

# Závěr

Tato práce se zabývá návrhem HW a FW bezdrátového lokalizačního vývojového tagu, založeného na technologii UWB, s ohledem na nízkou spotřebu. V první části práce byly vysvětleny základní principy bezdrátových lokalizačních metod s podrobným zaměřením na UWB. Dále byly popsány principy RTOS systémů a jejich úskalí. Byli popsáni tři kandidáti a z nich vybrán systém FreeRTOS.

V následující kapitole byla popsána specifikace, na základě které bylo navrženo blokové schéma obvodu. Dle specifikace a vybrané technologie byl proveden výběr a popis součástek s ohledem na minimální rozměry a nízkou spotřebu. Následovala implementace knihoven součástek pro Eagle CAD a vytvoření výsledného schématu s důrazem na budoucí rozšiřitelnost a snadnou testovatelnost. Následovalo rozmístění součástek s ohledem na výrobní technologii a signálovou a napájecí integritu. Deska plošných spojů byla manuálně osazena v prostorách laboratoří FIT VUT v Brně a následně oživena. Prototyp byl vytvořen úspěšně na první pokus.

Výsledný vývojový tag se skládá z UWB transceiveru DW1000, MCU NRF52832, akcelerometru, dvou IMU jednotek, magnetometru, barometru s teplotním čidlem, E-ink displeje, NFC EEPROM paměti, LEDek, tlačítek, baterie, nabíječe baterie, konektoru pro debugování a dalších nezbytných součástek. Je plně rozšiřitelný a testovatelný díky vyvedeným pin-headerům, test-pointům a rozšiřujícím periferiím na desce, které lze připojit na volné piny. Desku je možné testovat i v terénu díky konektoru na baterii a nabíjecímu obvodu.

Po HW implementaci následovala specifikace chování firmware s ohledem na nízkou spotřebu. Na základě znalosti RTOS byl iterativně vytvořen návrh s důrazem na maximální rozšiřitelnost, přenositelnost a čitelnost kódu. Při samotné implementaci byla odhalena mnohá úskalí, kupříkladu chybové IDE, nekompatibilní debugger a další. Byly využity technologie jako SEGGER RTT, SEGGER SystemView pro její debugování a CMSIS pro snadnější přenositelnost. Dále bylo vymyšleno a implementováno vlastní schéma přenositelnosti, snadného debugování napříč platformami a jednoduché konfigurace.

Byl vytvořen nízko-příkonový firmware, který pracuje na FreeRTOS systému. Nízkého příkonu dosahuje především použitím tickless módu. Aktuální verze firmware umožňuje odesílání blinků s různou periodou, výpis aktuálních údajů na E-Ink displej, snímání údajů ze senzorů a ovládání pomocí tlačítek. Firmware je možné snadno rozšířit následováním instrukcí a myšlenek uvedených v textu.

K závěru celé práce byl firmware otestován na proudovou spotřebu různých módů a stabilitu systému. Zjistilo se, že v nejnižším módu spotřeby může tag při frekvenci odesílání blinků 500 ms vydržet až 16 dní na baterii. Dále byly zjištěny doby běhů jednotlivých procesů s návrhem jejich snížení.

### Navrhovaná rozšíření

Vývojovou desku lze dále rozšiřovat díky možnosti připojení dalších externích součástek a používat ji pro budoucí vývoj a testování. Deska dále umožňuje použití součástek, přítomných jako rozšíření práce na desce. Jedná se o dvě IMU jednotky, magnetometr, akcelerometr a barometr s teploměrem. K těmto součástkám je možné napsat ovladače a zakomponovat je do stávajícího firmware s rozšířením jeho funkcionality, například odesíláním dat s aktuálním natočením a výškou tagu.

Další možností vylepšení je přepracování návrhu HW tagu. Mělo by dojít k zakomponování propojek, sloužících k odpojení/připojení napájení součástek přítomných na desce, aby nenarušovaly její testování (spotřeba, rušení).

Spotřebu tagu lze dále snižovat, například použitím DMA přenosu v případě objemnější datové komunikace s externími periferiemi. Zajímavým zjištěním by mohlo být nalezení optimálního bodu, kdy se DMA přenos vyplatí a kdy už nikoliv.

Další zajímavou myšlenkou je implementace algoritmu inverzního TDOA, kdy je tag aktivním příjemcem UWB signálu. Lokalizace pak probíhá na základě přesné znalosti poloh kotev.

# Literatura


[1] Alarifi, A., Al Salman, A., Alsaleh, M., Alnafessah, A., Alhadhrami, S. et al. Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances. *Sensors*. Květen 2016, roč. 16, s. 1–36.

[2] Dow, J. M., Neilan, R. E. a Rizos, C. The International GNSS Service in a changing landscape of Global Navigation Satellite Systems. *Journal of Geodesy*. Mar 2009, roč. 83, č. 3, s. 191–198. Dostupné z: https://doi.org/10.1007/s00190-008-0300-3. ISSN 1432-1394.

[3] Liu, H., Darabi, H., Banerjee, P. a Liu, J. Survey of Wireless Indoor Positioning Techniques and Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*. Nov 2007, roč. 37, č. 6, s. 1067–1080. ISSN 1558-2442.

[4] Shi, W., Du, J., Cao, X., Yu, Y., Cao, Y. et al. IKULDAS: An Improved kNN-Based UHF RFID Indoor Localization Algorithm for Directional Radiation Scenario. *Sensors*. Únor 2019, roč. 19, č. 4, s. 968.

[5] Networks, S. *Real-time location system (RTLS) for indoor tracking*. Jan 2019. [Online, navštíveno 2.11.2019]. Dostupné z: http://www.sewio.net/.

[6] Song, Z., Jiang, G. a Huang, C. A Survey on Indoor Positioning Technologies. In: Zhou, Q., ed. *Theoretical and Mathematical Foundations of Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 198–206. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-642-24999-0_28. ISBN 978-3-642-24999-0.

[7] Le Dortz, N., Gain, F. a Zetterberg, P. WiFi fingerprint indoor positioning system using probability distribution comparison. In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. March 2012, s. 2301–2304. ISSN 1520-6149.

[8] Zafari, F., Gkelias, A. a Leung, K. K. A Survey of Indoor Localization Systems and Technologies. *IEEE Communications Surveys Tutorials*. thirdquarter 2019, roč. 21, č. 3, s. 2568–2599. ISSN 2373-745X.

[9] Varshavsky, A., Lara, E. de, Hightower, J., LaMarca, A. a Otsason, V. GSM Indoor Localization. *Pervasive Mob. Comput.* NLD: Elsevier Science Publishers B. V. prosinec 2007, roč. 3, č. 6, s. 698–720. Dostupné z: https://doi.org/10.1016/j.pmcj.2007.07.004. ISSN 1574-1192.

[10] Moyer, B. *Dead reckoning error*. 2018. [Online, navštíveno 4.11.2019]. Dostupné z: https://www.insidetheiot.com/accels-and-gyros-better-together-with-gps/dead-reckoning-error/.

[11] Pandey, S. a Agrawal, P. A survey on localization techniques for wireless networks. *Journal of the Chinese Institute of Engineers*. Taylor & Francis. 2006, roč. 29, č. 7, s. 1125–1148. Dostupné z: https://doi.org/10.1080/02533839.2006.9671216.

[12] Networks, S. *Two Way Ranging (TWR)*. Aug 2019. [Online, navštíveno 2.11.2019]. Dostupné z: https://www.sewio.net/uwb-technology/two-way-ranging/.

[13] Pužmanová, R. *Ultra-širokopásmové sítě*. Internet Info Lupa.cz (www.lupa.cz) Server o českém Internetu, Apr 2002. [Online, navštíveno 15.11.2019]. Dostupné z: https://www.lupa.cz/clanky/ultra-sirokopasmove-site/.

[14] Pužmanová, R. *UltraWideBand*. Internet Info Lupa.cz (www.lupa.cz) Server o českém Internetu, Dec 2007. [Online, navštíveno 15.11.2019]. Dostupné z: https://www.lupa.cz/clanky/ultrawideband/.

[15] Dragomirescu, D., Kraemer, M., Jatlaoui, M., Pons, P., Aubert, H. et al. 60GHz Wireless Nano-Sensors Network for Structure Health Monitoring as Enabler for Safer, Greener Aircrafts. *Proceedings of SPIE - The International Society for Optical Engineering*. Září 2010, roč. 7821, s. 10.

[16] Ghavami, M. *Ultra Wideband Signals and Systems in Communication Engineering*. Wiley, jul 2004. ISBN 0470867515.

[17] Wikipedia contributors. *IEEE 802.15.4*. Wikimedia Foundation, May 2019. [Online, navštíveno 17.11.2019]. Dostupné z: https://en.wikipedia.org/wiki/IEEE_802.15.4.

[18] Group, W. W. P. A. N. W. W. IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*. Sep. 2011, s. 1–314. ISSN null.

[19] Karapistoli, E., Pavlidou, F., Gragopoulos, I. a Tsetsinas, I. An overview of the IEEE 802.15.4a Standard. *IEEE Communications Magazine*. January 2010, roč. 48, č. 1, s. 47–53. ISSN 1558-1896.

[20] Buttazzo, G. C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications (Real-Time Systems Series Book 24)*. Springer, sep 2011. ISBN 0792399943.

[21] Barry, R. *Mastering the FreeRTOS Real Time Kernel, A Hands-On Tutorial Guide*. 2016. [Online, navštíveno 16.11.2019]. Dostupné z: https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf.

[22] Phavorin, G., Richard, P., Goossens, J., Maiza, C., George, L. et al. Online and offline scheduling with cache-related preemption delays. *Real-Time Systems*. Jul 2018, roč. 54, č. 3, s. 662–699. Dostupné z: https://doi.org/10.1007/s11241-017-9275-6. ISSN 1573-1383.

[23] GmbH, S. M. *EmbOS & embOS-MPU, Real-Time Operating System, User Guide & Reference Manual*. Dec 2019. [Online, navštíveno 18.11.2019]. Dostupné z: https://www.segger.com/downloads/embos/UM01001.

[24] Walls, C. *Power management in embedded software*. Aug 2015. [Online, navštíveno 21.11.2019]. Dostupné z: https://www.embedded.com/power-management-in-embedded-software/.

[25] *Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions*. [Online, navštíveno 18.11.2019]. Dostupné z: http://www.freertos.org/.

[26] SEGGER. *EmbOS - RTOS, Real-Time Operating System*. 2019. [Online, navštíveno 12.11.2019]. Dostupné z: https://www.segger.com/products/rtos/embos/.

[27] Autodesk. *EAGLE: PCB Design Software*. 2019. [Online, navštíveno 29.11.2019]. Dostupné z: https://www.autodesk.com/products/eagle/overview.

[28] Lovejoy, B., Lovejoy, B., Lovejoy, B. a Eu. *U1 chip in iPhone 11 is Apple's own tech, not rebadged chip*. Oct 2019. [Online, navštíveno 24.10.2019]. Dostupné z: https://9to5mac.com/2019/10/11/u1-chip/.

[29] Morra, J. *NXP Looks to Enter New Era With Ultra-Wideband Chip*. 2019. [Online, navštíveno 24.10.2019]. Dostupné z: https://www.electronicdesign.com/technologies/analog/article/21808741/nxp-looks-to-enter-new-era-with-ultrawideband-chip.

[30] DECAWAVE. *DW1000 Radio IC*. 2019. [Online, navštíveno 24.10.2019]. Dostupné z: https://www.decawave.com/product/dw1000-radio-ic/.

[31] DECAWAVE. *Our Products*. 2019. [Online, navštíveno 5.10.2019]. Dostupné z: https://www.decawave.com/products/.

[32] *212x104, 2.13inch flexible E-Ink raw display panel*. [Online, navštíveno 13.2.2020]. Dostupné z: https://www.waveshare.com/product/displays/e-paper/epaper-3/2.13inch-e-paper-d.htm.

[33] *Embitz*. [Online, navštíveno 1.3.2020]. Dostupné z: https://www.embitz.org/.

[34] *Real-Time Transfer (RTT)*. [Online, navštíveno 1.11.2019]. Dostupné z: https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/.

[35] *NRF5 SDK*. [Online, navštíveno 3.11.2019]. Dostupné z: https://www.nordicsemi.com/Software-and-tools/Software/nRF5-SDK/Download.

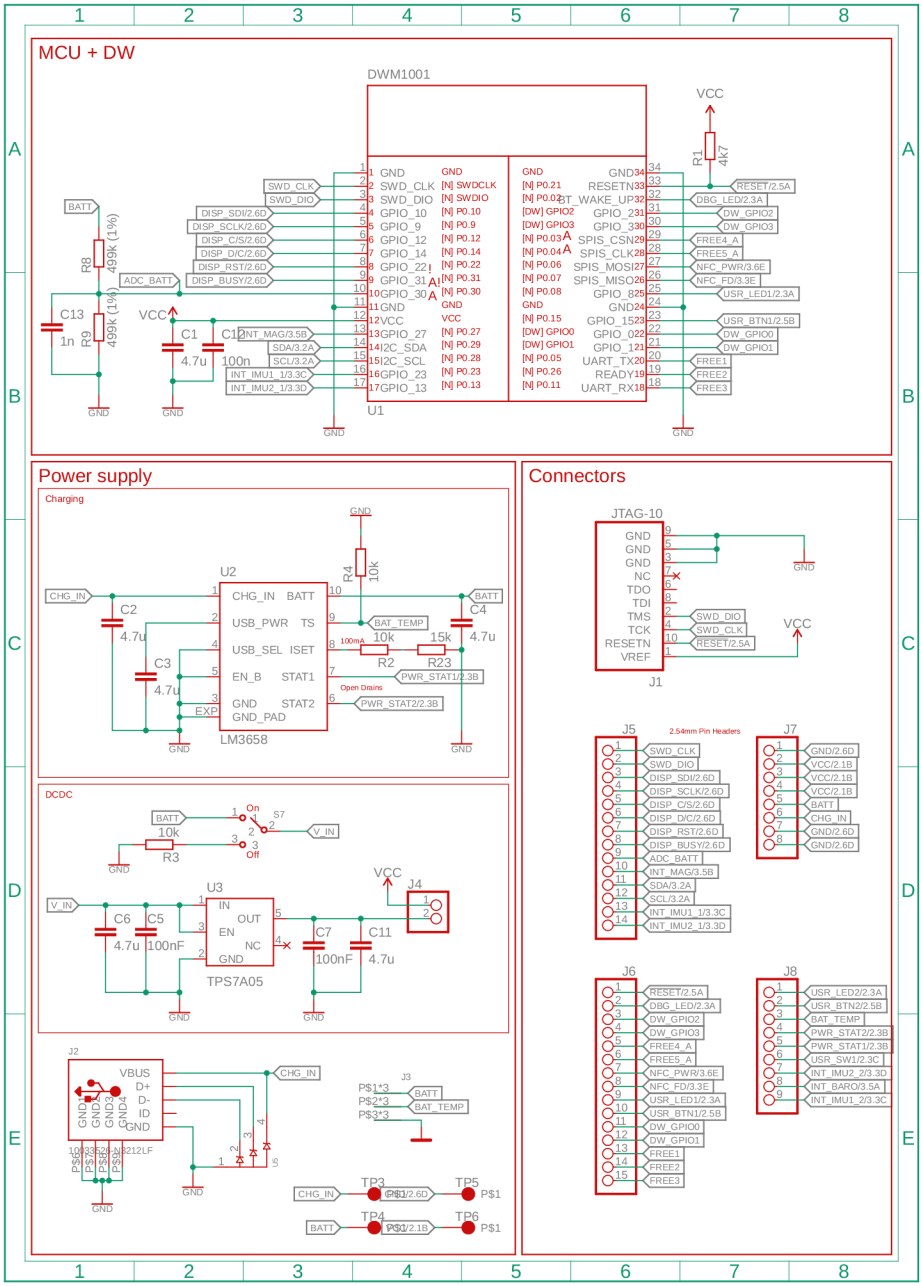
[36] Ltd, A. *CMSIS*. [Online, navštíveno 20.11.2019]. Dostupné z: https://developer.arm.com/tools-and-software/embedded/cmsis.

[37] *DW1000 Radio IC*. Jun 2020. [Online, navštíveno 20.10.2019]. Dostupné z: https://www.decawave.com/product/dw1000-radio-ic/.

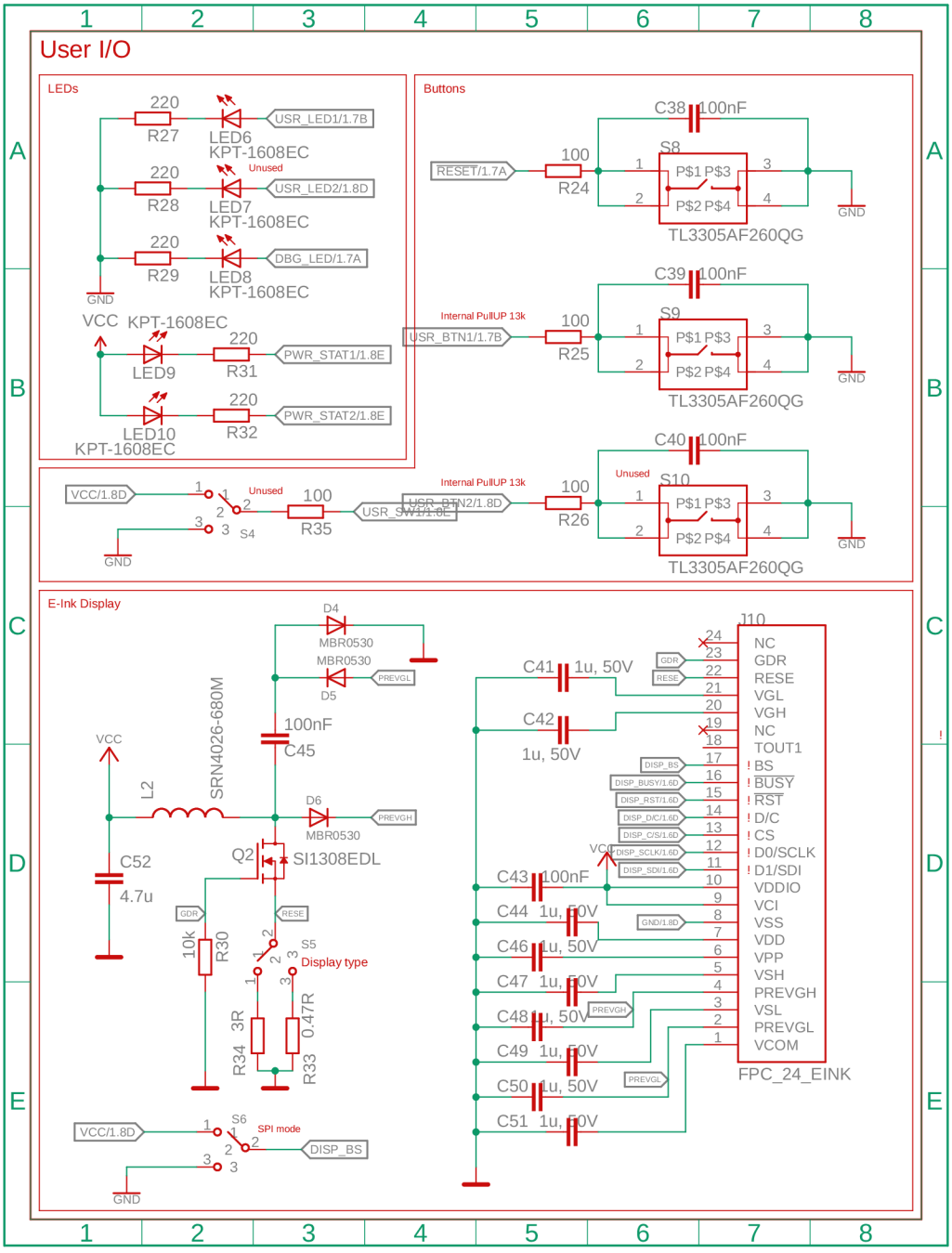
[38] *DWM1001 Development Board*. Aug 2019. [Online, navštíveno 6.12.2019]. Dostupné z: https://www.decawave.com/product/dwm1001-development-board/.

# Příloha A

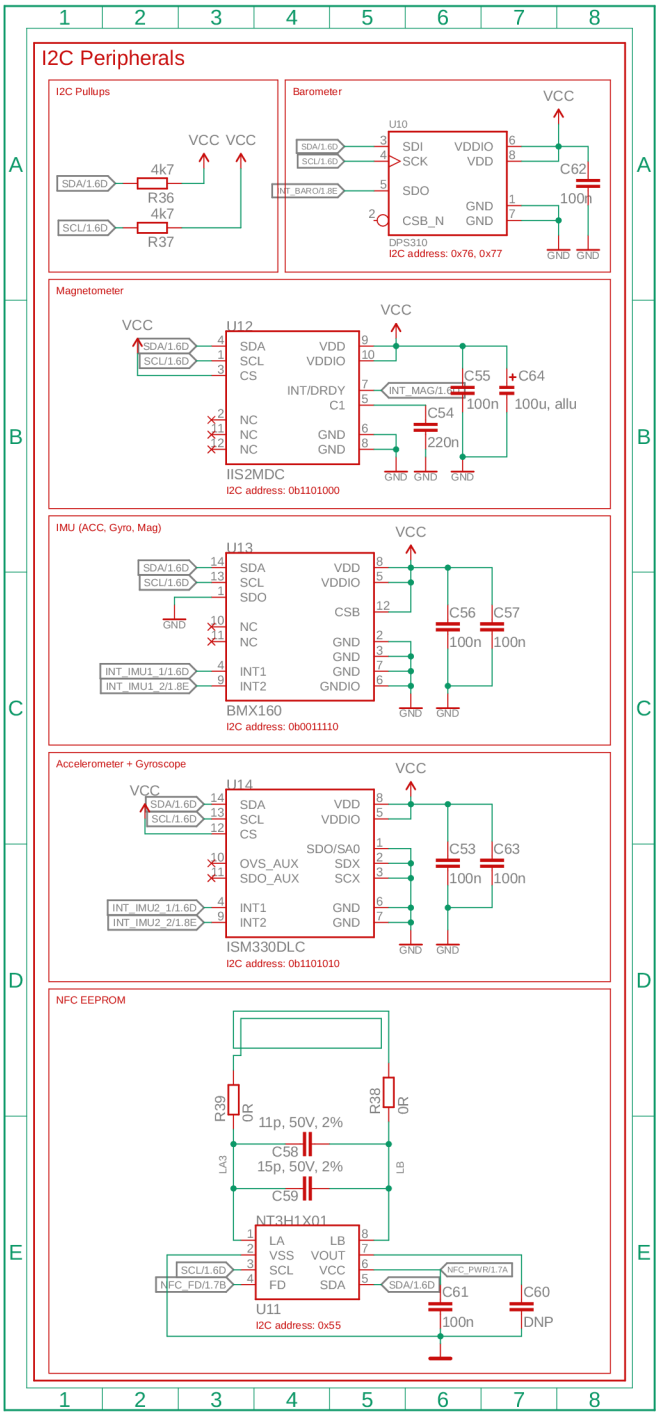
# Schéma zapojení




Obrázek A.1: Schéma zapojení, část 1 z 3




Obrázek A.2: Schéma zapojení, část 2 z 3




Obrázek A.3: Schéma zapojení, část 3 z 3

# Příloha B

# Kusovník DPS

| ID | Označení/hodnota | Typ pouzdra |
|---|---|---|
| U1 | DWM1001C | 20-SMD Module |
| U2 | LM3658SD-B, LLP10 | WSON, 3 x 3 mm |
| U3 | TPS7A05 | SOT23 (5) |
| U4 | DPS310 | 8-pin LGA, 2,0 x 2,5 mm |
| U5 | SP0503BAHTG | SOT-143 (4 piny) |
| U6 | IIS2MDC | LGA-12 |
| U7 | BMX160 | LGA-14 2,5 x 3,0 mm |
| U8 | ISM330DLC | LGA-14L |
| U9 | NT3H21112k EEPROM | TSSOP8 |
| S1, S2, S3 | TL3305AF260QG | SMD |
| S4, S5, S6, D7 | OS102011MS2QN1 | THT |
| Q1 | SI1308EDL-T1-GE3 | SC70 |
| D1, D3, D4 | MBR0530 | SOD123 |
| J1 | JTAG-10 | THT |
| J2 | USB MINI B | THT |
| J3 | 51201-3P | SMD |
| J4, J5 | Pin header, 17 pinů | 2,54 mm, THT |
| J6 | Pin header, 4 piny | 2,54 mm, THT |
| J7 | Pin header, 7 pinů | 2,54 mm, THT |
| LED(1, 2, 3, 4, 5) | Červená barva | SMD, 0603 |
| R1, R10, R11 | 4700 Ω | SMD, 0603 |
| R12, R13 | 0 Ω | SMD, 0603 |
| R14, R15, R16, R19, R20 | 220 Ω | SMD, 0603 |

Tabulka B.1: Použité součástky (U – integrovaný obvod, S – přepínač, Q – tranzistor, D – dioda, J – konektor, R – rezistor, C – kondenzátor, L – cívka)

| ID | Označení/hodnota | Typ pouzdra |
|---|---|---|
| R18 | 0,47 Ω | SMD, 0603 |
| R2 | 22K Ω | SMD, 0603 |
| R31 | 3 Ω | SMD, 0603 |
| R4, R3, R17 | 10k Ω | SMD, 0603 |
| R5, R6, R7 | 100 Ω | SMD, 0603 |
| R8, R9 | 499k Ω(1%) | SMD, 0603 |
| C11, C1, C2, C3, C4, C6, C35 | keramický, 4,7 uF | SMD, 0603 |
| C12, C8, C9, C10, C36, C16, C17, C18, C14, C37, C23, C5, C7, C26, C34 | keramický, 100 nF | SMD, 0603 |
| C13 | keramický, 1 nF | SMD, 0603 |
| C15 | keramický, 220 nF | SMD, 0603 |
| C19 | hliněný, 10uF, ESR maximálně 200M Ω | SMD, 0603 |
| C20 | 11 pF, 50 V, 2% | SMD, 0603 |
| C21 | 15 pF, 50 V, 2% | SMD, 0603 |
| C24, C25, C27, C28, C29, C30, C31, C32, C33 | 1 uF, 50 V, keramický | SMD, 0603 |
| L1 | 68 μH | |

Tabulka B.2: Použité součástky (U – integrovaný obvod, S – přepínač, Q – tranzistor, D – dioda, J – konektor, R – rezistor, C – kondenzátor, L – cívka)

# Příloha C

# Obsah přiloženého paměťového média

Přiložené paměťové médium obsahuje:

- ./RtosTag/ - adresář s implementací projektu,
- ./doc/ - adresář se zdrojovými kódy této technické zprávy,
- ./technickaZprava.pdf - tuto technickou zprávu ve formátu PDF.

# Příloha D

# Souborová struktura implementace projektu RtosTag

Projekt RtosTag obsahuje následující soubory a složky (pro zjednodušení nejsou uvedeny veškeré zdrojové soubory, ale pouze složky, které je sdružují):

```
RtosTag/                          # hlavní složka projektu
  ./README.txt                    # základní instrukce ke zprovoznění projektu
  ./HW/                           # složka obsahující HW implementaci
    ./RtosTag.sch                 # schéma HW zapojení tagu
    ./RtosTag.brd                 # layout HW zapojení tagu
    ./Libraries/Devices.lbr       # knihovny součástek
  ./FW/                           # složka obsahující SW implementaci
    ./RtosTag.ebp                 # soubor projektu, kompatibilní s IDE Embitz
    ./main.c                      # hlavní zdrojový soubor projektu
    ./app_specific/               # složka s aplikačně specifickými zdrojovými kódy
      ./button/                   # implementace knihovny tlačítek
      ./charging/                 # implementace detekce nabíjení tagu
      ./config/                   # implementace uživatelských konfigurací tagu
      ./display/                  # implementace vykreslování na displej
      ./eeprom/                   # zjednodušení ovládání EEPROM paměti
      ./sensors/                  # implementace senzorů
      ./tag_behavior/             # implementace jádra chování
        ./app_behav_events.c/h    # implementace událostí
        ./app_behav_states.c/h    # implementace stavů
        ./app_behav_tasks.c/h     # implementace tasků
      ./uwb/                      # implementace správy a odesílání UWB zpráv
      ./app_tag_errors.c/h        # implementace reakcí na chyby
      ./app_tag_runtime.c/h       # implementace globálních dat
      ./app_tag_settings.c/h      # implementace nastavení tagu
    ./common/                     # implementace přenositelných souborů
      ./ports/                    # implementace přenositelnosti, pomocí portů
        ./com_port_global.h       # implementace přenositelnosti, hlavní soubor
        ./com_port_nrf52832.h     # implementace přenositelnosti pro NRF52832
      ./sewio/                    # implementace kompatibility s firmou
      ./thesis/                   # implementace globálních funkcí v této práci
```

```
./config/SDK/                # konfigurace SDK
  ./sdk_config.h             # konfigurace SDK
  ./app_config.c/h           # přidaná uživatelská konfigurace SDK
./config/tag/                # konfigurace tagu
  ./cnf_global.c/h           # všechny globální konfigurace
  ./cnf_global_pins.h        # globální konfigurace pinů
  ./cnf_global_debug.h       # globální konfigurace logování
  ./cnf_global_config.h      # globální konfigurace tagu
./drivers/                   # implementace ovladačů
  ./bindings/                # platformě závislý kód ovladačů
  ./DW1000/                  # ovladač k DW1000
  ./EPD_Waveshare_EINK/      # ovladač k displeji
  ./NRF52temp/               # ovladač k teplotnímu čidlu
  ./NT3H1X01/                # ovladač k NFC EEPROM paměti
./libraries/                 # implementace knihoven
  ./CRC/                     # knihovna k CRC součtům
  ./LED/                     # knihovna k ovládání LEDek
  ./random/                  # knihovna k náhodnému generování čísel
./linker/                    # linker soubory
./startup/                   # startup zdrojové soubory
./third_parties/             # knihovny třetích stran
  ./FreeRTOS/                # FreeRTOS v10.0.x
    ./config/                # konfigurace FreeRTOSu
  ./SystemView_Src_V252d/    # SEGGER RTT a SystemView
    ./Config/                # konfigurace RTT
```

# Příloha E

# Návod na zprovoznění

Pro modifikaci a debugování projektu je třeba následovat tyto kroky:

1. stáhnout IDE Embitz, ve verzi 1.1 [33],
2. stáhnout SDK výrobce, ve verzi 15.3.0 [35] a rozbalit jej do nadsložky souboru projektu RtosTag.ebp,
3. v IDE Embitz otevřít soubor projektu RtosTag.ebp,
4. připojit kompatibilní debugger, například J-Link Edu,
5. připojit debugger k vývojové desce, pomocí pinů SWDIO, SWDCLK, RESET, VCC a GND,
6. nyní lze modifikovat nastavení projektu, debugovat a nahrávat nový firmware.