Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Vývoj periferie s využitím Bluetooth Low Energy SoC

Bakalářská práce

Autor: Martin Donát
Studijní obor: AI

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové duben 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 17.4.2018 Martin Donát

Poděkování:

Chtěl bych poděkovat vedoucímu bakalářské práce Ing. Pavlu Křížovi, Ph.D. za velice cenné rady, které mě vždy navedly správným směrem.

Anotace

Bakalářská práce se věnuje seznámení s Bluetooth Low Energy technologií. Dále je popsán vývojový kit CC2650 Sensor Tag a také způsob realizace technologie Bluetooth Low Energy na tomto kitu. Cílem bakalářské práce bylo vytvořit přenosné zařízení sloužící pro experimenty se zpřesňováním indoor lokalizace pomocí platformy System-on-Chip CC2650 Sensor Tag od Texas Instruments. Dále je v textu vysvětlen princip komunikace mezi Sensor Tagem a smartphonem. Poté jsou zde popsány nezbytné služby a charakteristiky pro komunikaci. V bakalářské práci je také detailně popsáno, jak bylo dané přenosné zařízení vytvořeno. Prototyp přenosného zařízení dokáže uložit rádiové otisky od 256 různých zařízení, přičemž celkový počet uložených rádiových otisků může být až 1700.

Annotation

Title: Development of Bluetooth Low Energy Peripheral based on a SoC

Bachelor thesis deals with the introduction of Bluetooth Low Energy technology. The development kit CC2650 Sensor Tag is also described, as well as the implementation of Bluetooth Low Energy on this kit. The aim of this bachelor thesis was to create a portable device used for indoor location refinement experiments using the Texas Instruments System-on-Chip CC2650 Sensor Tag platform. Furthermore, the principle of communication between Sensor Tag and smartphone is explained in the text. Then the necessary services and communication characteristics are described. The bachelor thesis also describes in detail how the portable device was created. The portable device prototype can store radio imprints from 256 different devices, with a total number of stored radio imprints up to 1700.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V dnešní době s rozvojem technologií vzniká nový trend Internet of Things, kde v popředí stojí technologie jako Bluetooth, Wi-Fi a další podobné technologie. Tyto bezdrátové technologie slouží pro komunikaci mezi elektronickými zařízeními a umožňují tak uživateli například ovládat svůj domov z chytrého telefonu. V současnosti je technologie Bluetooth součástí téměř každého elektronického zařízení. Je tedy zřejmé, že tato technologie má velký význam i do budoucna.

Bakalářská práce se bude zabývat analýzou možností vývoje Bluetooth Low Energy periferie, pomocí řešení System-on-Chip CC2650 Sensor Tag od Texas Instruments. Cílem bude navrhnout a implementovat konkrétní aplikaci periferie komunikující se smartphonem. Úkolem periferie bude naskenovat okolní iBeacon zařízení, přičemž periferie uloží informace o těchto naskenovaných iBeacon zařízeních a umožní smartphonu tato data vyčíst. Skenování periferie bude iniciováno ze smartphonu.

Tato periferie by vzhledem k malým rozměrům mohla být využita například jako náramek pro seniory. Periferie by sledovala v domě seniorovu lokaci, a pokud by daná osoba zůstala dlouho bez pohybu například v koupelně, dalo by se na tuto situaci zareagovat.

2 Cíl práce

Cílem bakalářské práce je analyzovat možnosti vývoje Bluetooth Low Energy periferie a vytvořit prototyp přenosného zařízení sloužícího pro experimenty se zpřesňováním indoor lokalizace. Jako periferie bylo zvoleno System-on-Chip řešení od Texas Instruments nacházející se na vývojovém kitu CC2650 Sensor Tag. Toto prototypové zařízení bude přijímat pakety vysílané zařízeními iBeacon. Veškeré zachycené pakety tímto zařízením budou uloženy a zpřístupněny skrze definované charakteristiky. Toto prototypové zařízení bude komunikovat primárně se smartphonem, pomocí kterého bude zahájeno skenování okolních iBeacon zařízení. Skenování bude trvat po určitou dobu podle hodnoty zapsané smartphonem do charakteristiky prototypu zařízení. Po dokončení skenu smartphone skrze charakteristiky vyčte všechny zachycené pakety a uloží je.

3 Technologie Bluetooth Low Energy (BLE)

3.1 Bluetooth

Bluetooth je bezdrátová technologie, standardizovaná jako IEEE 802.15.1, sloužící pro přenos dat mezi zařízeními na krátké vzdálenosti. Byla vyvinuta, aby nahradila kabelové spojení přenosných i pevných zařízení. Tuto technologii spravuje organizace Bluetooth Special Interest Group (SIG), která se skládá z více než 30 000 společností.

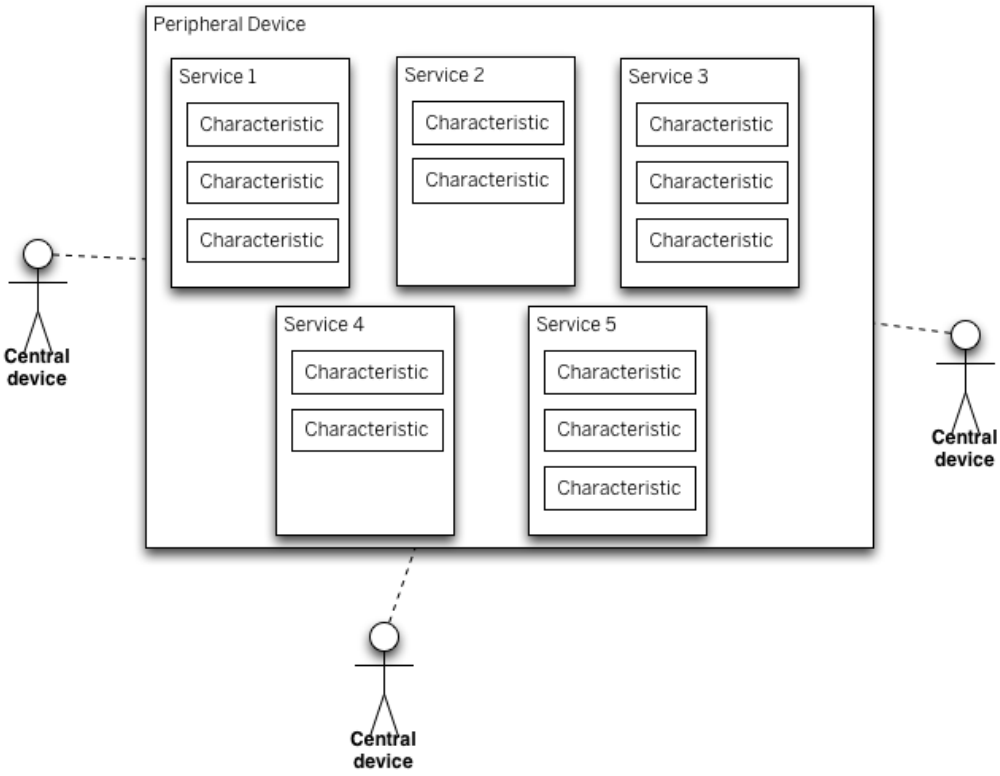
Bluetooth využívá bezlicenčního pásma 2,4 GHz rozděleného do 79 kanálů s rozestupem 1 MHz. Komunikace probíhá na všech kanálech s využitím technologie Frequency Hopping Spread Spectrum (FHSS), kdy se během komunikace několikrát změní vysílání dat na jinou frekvenci. Tato technologie se snaží zvýšit počet souběžně komunikujících zařízení a eliminovat rušení na bezlicenčním pásmu 2,4 GHz, na kterém pracují i jiné technologie, například Wi-Fi. Na jakou frekvenci se přeladí vysílání je dáno pseudo-náhodným generátorem, který generuje zařízení označené jako master. Zbylá zařízení, která jsou k tomuto zařízení připojena, se nazývají slave [2].

Bluetooth má dvě varianty - Bluetooth BR/EDR známé jako Bluetooth Classic a Bluetooth with Low Energy functionality (LE). V dnešních zařízeních, hlavně telefonech a tabletech, najdeme integrovaný obvod s duálním režimem, kde jsou podporovány obě varianty Bluetooth [1].

3.2 Bluetooth Low Energy

Od verze Bluetooth 4.0 přijala organizace SIG specifikaci Bluetooth Low Energy, dále BLE. Tato technologie je zaměřena na nízkou energetickou spotřebu a pro přenos menšího objemu dat. Díky optimalizaci mohou zařízení s BLE být velice malé a pro napájení si vystačí s lithiovou baterii ve velikosti mince. Pokud zařízení nevysílá signál, tak je v úsporném režimu. Spojení mezi zařízeními trvá pouze pár milisekund oproti klasickému Bluetooth, kde spojení může trvat až 100 ms. BLE využívá též bezlicenčního pásma 2,4 GHz, ale využívá pouze 40 kanálů s rozestupem 2 MHz [2].

BLE zařízení jsou rozdělena do dvou skupin – central devices a peripheral devices. Pokud tato zařízení porovnáme s modelem klient-server, tak peripheral device je server, který poskytuje určité služby. Central device je klient, který čte a případně přepisuje data serveru. Komunikace zařízení BLE se dá přirovnat ke komunitní nástěnce, jak je znázorněno na Obr. 1. Zařízení, která čtou data ze senzorů a v případě změny dat změní data na nástěnce, se nazývají peripheral devices. Ostatní zařízení, která data z nástěnky čtou, se nazývají central devices. Central device může data z nástěnky přečíst kdykoliv potřebuje, například pravidelným časovačem. Tento způsob však není příliš energeticky úsporný. BLE dále implementuje mechanismus notify, který umožní peripheral device poslat data central device, pokud se určitá data změní [4].



Obr. 1 Komunikace mezi central a peripheral devices. Zdroj [4]

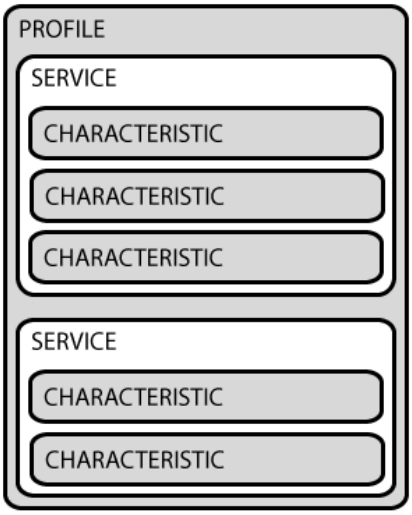
Vzhledem k zaměření BLE na nízkou energickou spotřebu je zařízení schopno dosáhnout přenosové rychlosti do 2 Mb/s. Tato rychlost není příliš velká oproti klasickému Bluetooth, kde přenosová rychlost může dosahovat až 54 Mb/s se standardem 802.11 AMP. Proto se zařízení s BLE nehodí například pro

streamování hudby, kde je potřeba přenést větší objem dat. Naopak se skvěle hodí pro fitness zařízení, myši, klávesnice a další zařízení, kde není potřeba přenést velké množství dat, ale je očekávána dlouhá životnost baterie [2].

3.2.1 Služby, Charakteristiky, Profily

Služby rozdělují jednotlivá elementární data do logických entit a dále obsahují specifické bloky dat zvané charakteristiky. Několik služeb je možné seskupit do profilu, jak je znázorněno na Obr. 2.

V následujícím příkladu jsou použity dvě služby. První služba ovládá klasickou LED diodu a bude mít dvě charakteristiky. První bude určovat, jestli LED dioda svítí a druhá bude určovat jas LED diody. Další služba bude ovládat RGB LED diodu a bude mít navíc charakteristiku, která bude určovat barvu. Tyto dvě služby jsou obdobné a je tedy vhodné je logicky seskupit do jednotného profilu.



Obr. 2 Znázornění uspořádání služeb, charakteristik a profilů. Zdroj [3]

Aby spolu dvě spojená zařízení mohla komunikovat, musí obě podporovat danou službu. Existují standardní služby[1], které je možné libovolně využívat a vlastní služby, které lze libovolně definovat. Pokud tedy začneme pracovat s BLE, je nutné se rozhodnout, zda využijeme některou ze standardních služeb nebo definujeme vlastní službu. Služby jsou identifikovány pomocí unikátního UUID

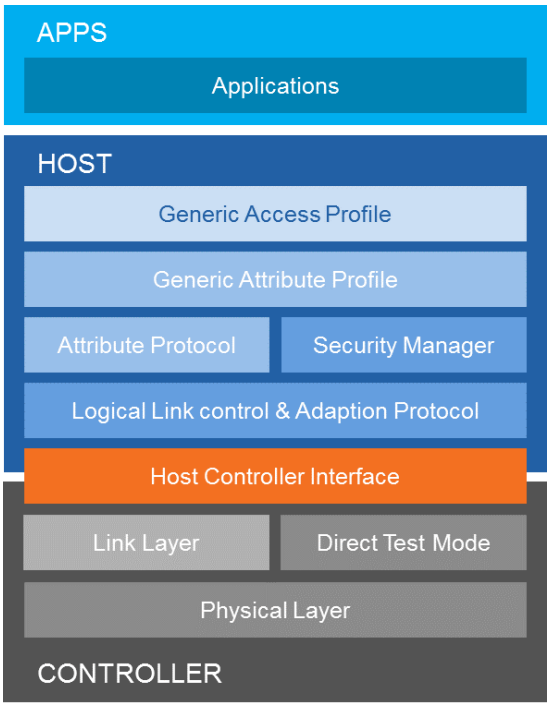
[1] https://www.bluetooth.com/specifications/gatt/services

(Universally Unique Identifier). Standardní služby mají 16 bitové UUID a vlastní služby mají 128 bitové UUID [3].

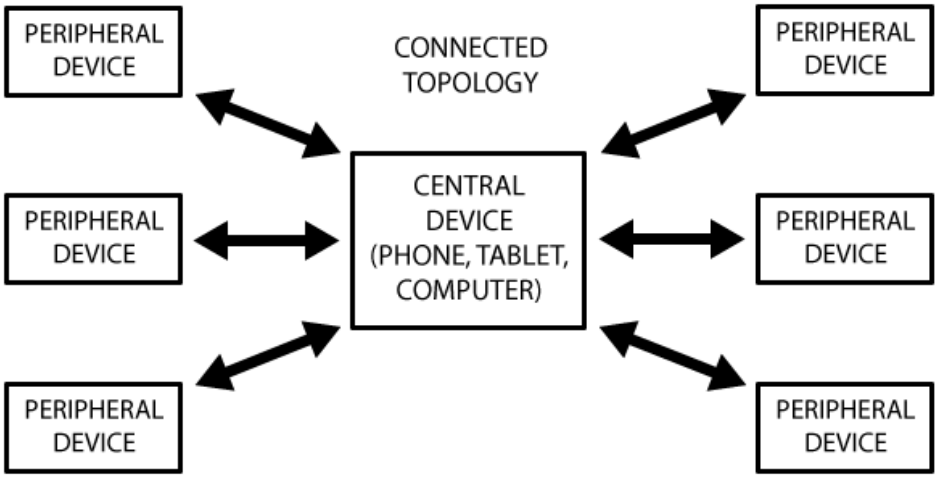
3.2.2 GATT

GATT je zkratka z anglického Generic Attribute Profile. BLE pracuje na několika vrstvách, viz Obr. 3. Jednou z těchto vrstev je i GATT. GATT je vrstva využívající vrstvu ATT (Attribute Protocol), skrze kterou je schopna definovat služby a charakteristiky. Dále GATT umožňuje přistupovat k jednotlivým službám a charakteristikám pomocí operací read, write, notify a indicate.

GATT se využívá především v BLE a ojediněle také v Bluetooth classic. GATT se začne používat až po vrstvě GAP (Generic Access Profile), která zprostředkovává spojení mezi dvěma zařízeními. BLE peripheral device může být připojeno pouze k jednomu central device, kdy central device může být například telefon, tablet atd. Naopak central device může být připojeno k více různým peripheral device zároveň. Tento fakt znázorňuje Obr. 4. Peripheral device je běžně GATT server, který poskytuje služby a charakteristiky, zatímco central device je GATT client, který iniciuje komunikaci a posílá požadavky na GATT server [3].



Obr. 3 Jednotlivé vrstvy BLE. Zdroj [1]

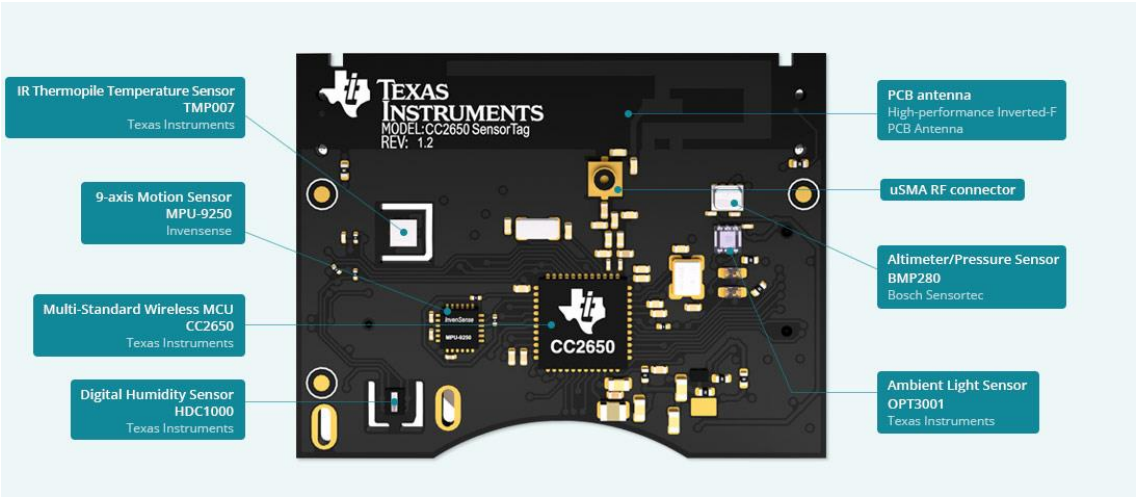


Obr. 4 Topologie mezi central a peripheral devices na úrovni GATT. Zdroj [3]

4 Implementace BLE na platformě Texas Instruments Simplelink

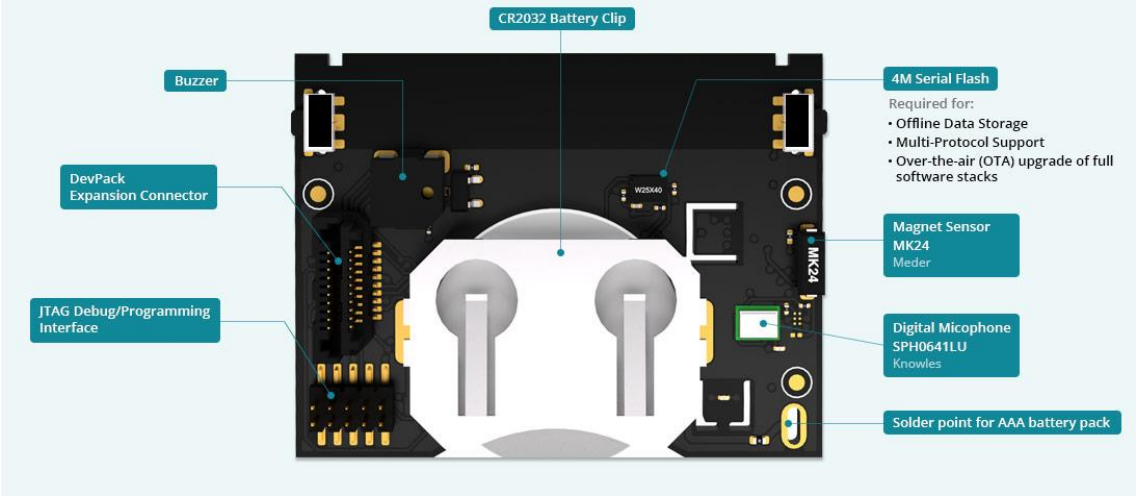
4.1 CC2650 Sensor Tag

Jedná se o multi-standardní a vysoce úsporný vývojový kit vhodný pro realizaci Internet of Things (IoT). Vývojový kit je napájen jednou baterií velikosti mince – CR2032, přičemž výrobce udává dobu životnosti této baterie v letech. Rozměry tohoto vývojového kitu jsou 5x6,7x1,4 cm (VxŠxH). Srdcem vývojového kitu je bezdrátový mikrokontroler CC2650, který podporuje standardy Bluetooth Low Energy, ZigBee a 6LoWPAN. Tento vývojový kit zahrnuje 10 MEMS[2] senzorů s nízkou energetickou náročností. Jedná se o následující senzory: senzor ambientního světla, digitální mikrofon, magnetický senzor, senzor vlhkosti a tlaku, akcelerometr, gyroskop, magnetometr, infračervený teploměr a teploměr okolní teploty. Dále jsou v tomto kitu integrovány dvě LED diody, dvě tlačítka, piezo měnič, mikro SMA konektor pro připojení externí antény, DevPack rozšiřující konektor a JTAG konektor pro připojení debuggeru [5].



Obr. 5 Přední strana vývojového kitu CC2650 Sensor Tag. Zdroj [6]

[2] Micro-Electro-Mechanical-Systems

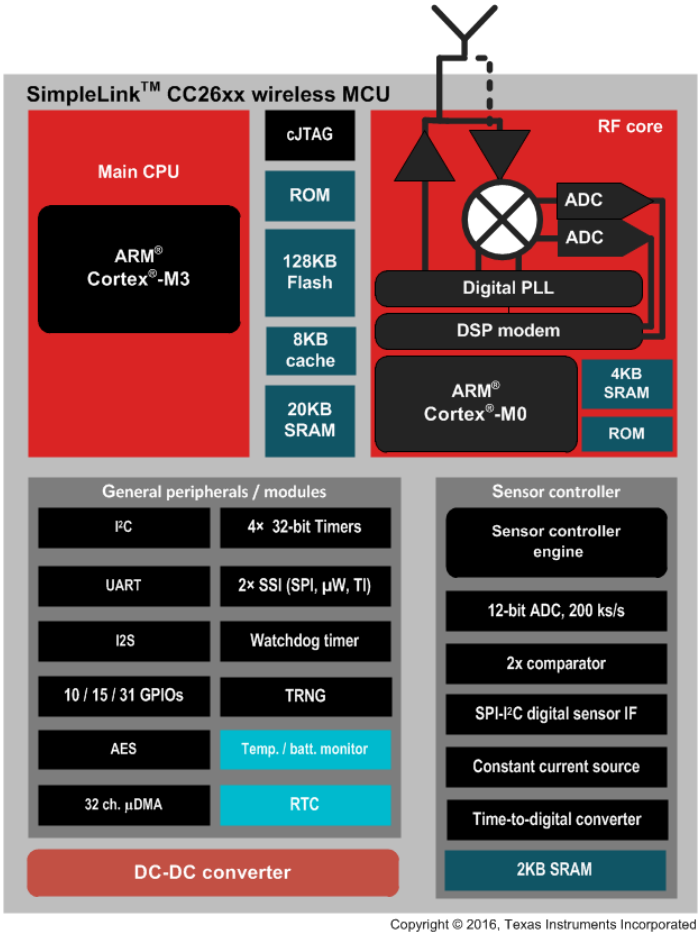


Obr. 6 Zadní strana vývojového kitu CC2650 Sensor Tag. Zdroj [6]

4.1.1 CC2650F128

Jak již bylo zmíněno výše v textu, bezdrátový mikrokontroler CC2650F128 je nedílnou součástí vývojového kitu CC2650 Sensor Tag. V následující podkapitole je popsán právě tento bezdrátový mikrokontroler, dále jen MCU.

CC2650 je bezdrátový MCU z rodiny CC26xx, která je cenově dostupná a zaměřená na nízkou energetickou spotřebu. Jedná se o bezdrátový MCU zaměřený na Bluetooth Low Energy, ZigBee, 6LoWPAN a ZigBee RF4CE. Skládá se ze dvou ARM procesorů. První z procesorů je 32 bitový ARM Cortex-M3 běžící na frekvenci 48 MHz, který je označovaný jako „Main CPU“. Na tomto procesoru běží vlastní aplikace. Druhý procesor je ARM Cortex-M0, který se stará o protokoly bezdrátové komunikace. Tento procesor není možné programovat. Oddělená architektura zvyšuje celkový výkon systému, snižuje energetickou náročnost a náročnost na flash paměť pro aplikaci. Dále je v CC2650 integrováno velké množství periferií např. sběrnice I2C, UART, I2S, AES-128 bezpečnostní modul, Real-Time Clock (RTC), čtyři 32 bitové časovače apod. včetně úsporného sensor controlleru. Sensor controller se dá výhodně využít jako rozhraní pro externí analogové i digitální senzory. Výhodou tohoto sensor controlleru je, že může autonomně sbírat data ze senzorů i když je zbytek systému v pohotovostním režimu. Vnitřní uspořádání CC2650 je znázorněno v blokovém schématu na Obr. 7 [7].



Obr. 7 Funkční blokové schéma bezdrátového MCU CC2650. Zdroj [7]

Uvnitř CC2650 je k dispozici 128 kB programovatelné nevolatilní flash paměti, která je určená pro uchovávání kódu a dat. Další pamětí, která se nachází uvnitř CC2650 je 20 kB statická RAM (SRAM). Tato paměť může být využita pro uložení dat i pro uložení vykonávaného kódu. Zároveň je tato paměť rozdělena do dvou 4 kB bloků a do dvou 6 kB bloků paměti. Zda si SRAM uchová data v pohotovostním režimu se dá povolit nebo zakázat pro každý blok této paměti zvlášť. Tímto nastavením se dá výhodně snížit energetická náročnost CC2650. Pokud je zakázaná flash cache, může být využita 8 kB cache jako univerzální RAM paměť. Paměť ROM obsahuje TI-RTOS[3] kernel, driverlib, nižší vrstvy bezdrátového protokol stacku a bootloader [8].

[3] Texas Instruments Real Time Operating System

4.2 Implementace BLE

Nyní následuje představení implementace technologie BLE na platformě Texas Instruments Simplelink. Samotná technologie Bluetooth je na této platformě realizována hardwarově pomocí bezdrátového MCU, jak již bylo zmíněno výše, a softwarově pomocí BLE stacku vytvořeného společností Texas Instruments. Tento BLE stack poskytuje API (Application Programming Interface), které umožňuje programování aplikací využívajících Bluetooth. Je tedy nezbytné mít BLE stack přítomný v takovéto aplikaci. Texas Instruments zvolil architekturu dvou samostatných projektů pro aplikace využívající Bluetooth. V jednom z těchto projektů se nachází samostatný BLE stack připravený od společnosti Texas Instruments. Ve druhém projektu je již samotná aplikace. Vzhledem k této architektuře je nutné zkompilovat zdrojový kód a následně jej nahrát do paměti MCU nejprve u projektu s BLE stackem. Poté již není nutné kompilovat a nahrávat zdrojový kód BLE stacku do MCU. Texas Instruments zvolil architekturu dvou projektů, aby bylo možné aplikaci aktualizovat nezávisle na BLE stacku. Dále Texas Instruments připravil ukázkové projekty aplikací s využitím Bluetooth, ve kterých se již nachází dva samostatné projekty podle výše zmíněné architektury. Texas Instruments doporučuje zvolit vhodný ukázkový projekt a na základě tohoto projektu vyvinout vlastní aplikaci. V této práci je aplikace založena na ukázkovém projektu simple_peripheral_observer.

4.2.1 Čtení a zapisování dat profilu

Komunikace mezi Sensor Tagem a ostatními zařízeními probíhá skrze BLE stack. Pokud zařízení požádá Sensor Tag o data konkrétní charakteristiky nebo chce data v charakteristice přepsat, musí BLE stack na tuto událost zareagovat. K tomuto účelu BLE stack využívá tzv. callback funkce. Callback funkce je funkce, kterou jedna entita předá druhé entitě, od které chce být v případě vzniku konkrétní situace kontaktována. V této callback funkci je pak definováno, jak se má zareagovat na vzniklou událost. BLE stack využívá dvě callback funkce pro kontaktování profilu v případě požadavku na data. V každém profilu tedy musí být tyto dvě callback funkce implementovány.

4.2.2 Read callback funkce

Tento callback slouží, jak již název napovídá, pro čtení dat. Když BLE stack přijme požadavek od GATT klienta na data určité charakteristiky, tak nejprve zkontroluje oprávnění, zda je daná charakteristika určená pro čtení. Pokud je daná charakteristika určená pro čtení, BLE stack zavolá read callback funkci profilu, ve kterém se nachází daná charakteristika. V této read callback funkci dojde primárně k nakopírování hodnoty dané charakteristiky na adresu v paměti, která přijde se zavoláním callback funkce BLE stackem. BLE stack poté tuto nakopírovanou hodnotu pošle GATT klientovi, který o ni požádal. Dále v této callback funkci může dojít k dalším různým zpracováním specifickým pro daný profil, případně může profil kontaktovat aplikaci o této události prostřednictvím aplikačního callbacku [13].

4.2.3 Write callback funkce

U write callback funkce je průběh obdobný jako u výše popsané read callback funkce. Tedy BLE stack přijme požadavek od GATT klienta na zápis dat do určité charakteristiky. Poté BLE stack zkontroluje oprávnění pro zápis. V případě, že je daná charakteristika určena pro zápis, BLE stack zavolá write callback funkci daného profilu. Při volání callback funkce BLE stack předá i adresu paměti, na které se nachází nová hodnota přijatá od GATT klienta. V této callback funkci dojde k validaci zapsané hodnoty, jako například zda není délka hodnoty v bajtech větší, než délka hodnoty uložené v charakteristice. Pokud je hodnota validní, dojde k nakopírování této hodnoty namísto původní hodnoty charakteristiky. Dále je možné opět provést v tomto callbacku další zpracování specifické pro daný profil, případně zavolat callback aplikace a tím ji kontaktovat o této události [13].

5 Analýza a návrh řešení

5.1 Analýza

Záměrem je vytvoření funkčního prototypu přenosného zařízení sloužícího pro experimenty se zpřesňováním indoor lokalizace. Prototyp přenosného zařízení bude zhotoven pomocí vývojového kitu CC2650 Sensor Tag, dále jen Sensor Tag. Toto zařízení umožní ve spolupráci s chytrým mobilním telefonem skenovat okolní iBeacon zařízení sloužící pro indoor lokalizaci a uložit o těchto zařízeních informace jako tzv. rádiové otisky. Rádiové otisky získané chytrým telefonem z tohoto přenosného zařízení budou využity pro experimenty se zpřesňováním indoor lokalizace.

5.1.1 Rádiový otisk

Jedná se o záznam informací, které získá Sensor Tag během skenování iBeaconů. Tento záznam informací bude obsahovat MAC adresu daného iBeaconu, RSSI (Received Signal Strength Indication) a časové razítko tzv. timestamp, kdy byl záznam pořízen.

5.1.2 Princip činnosti

Celá komunikace začne navázáním spojení mezi chytrým telefonem a Sensor Tagem. Sensor Tag bude poskytovat dvě služby. Pomocí první služby se zahájí skenování okolních iBeacon zařízení. Druhá služba poskytne rádiové otisky získané během skenování okolních iBeaconů.

Po navázání spojení chytrý telefon zapíše do charakteristiky první služby určitou hodnotu. Na základě zapsané hodnoty zahájí Sensor Tag skenování okolních iBeacon zařízení. Skenování probíhá po určitou dobu, podle zapsané hodnoty charakteristiky první služby. Po uplynutí doby se skenování ukončí a dojde k naplnění charakteristik druhé služby rádiovými otisky získanými během skenování. Chytrý telefon vyčte veškeré získané rádiové otisky pomocí charakteristik druhé služby. Celá komunikace je znázorněna na Obr. 8.

Obr. 8 Znázornění komunikace smartphonu se zařízením CC2650 Sensor Tag a okolních iBeaconů. Zdroj vlastní tvorba podle [9], [10], [11], [12]

5.2 Návrh řešení

Vzhledem k využití technologie Bluetooth a způsobu komunikace mezi bluetooth zařízeními, je nutné se zamyslet nad jednotlivými službami a charakteristikami. Důležitým faktorem pro tvorbu služeb a charakteristik jsou samotná data, která si zařízení potřebují vyměnit. V případě, že spolu více služeb souvisí je vhodné je seskupit do jednoho profilu.

5.2.1 Služby a charakteristiky

První služba se bude starat o zahájení skenování okolních iBeaconů a bude označená jako Beacons discovery. Tato služba bude obsahovat charakteristiku Start scan umožňující čtení a zápis. Pokud je do této charakteristiky zapsána hodnota větší než nula, Sensor Tag zahájí skenování na dobu, podle hodnoty přijaté v této charakteristice. Zapsaná hodnota v charakteristice setrvá, dokud skenování neskončí. Poté se charakteristika nastaví opět na výchozí hodnotu, kterou je nula. Čtení této charakteristiky umožní zjistit, zda Sensor Tag ještě skenuje.

Druhá služba bude poskytovat rádiové otisky získané během skenování iBeaconů a bude označená jako Beacons list. V této službě se bude nacházet několik charakteristik. Charakteristika Set index bude sloužit pro zápis indexu požadovaného rádiového otisku. Indexovat se bude od nuly do hodnoty

charakteristiky *Total count - 1*. Tato charakteristika bude vždy po dokončení skenování nastavena zpět na nulu. Všechny následující charakteristiky budou umožňovat pouze čtení. Charakteristika Total count bude vracet celkový počet získaných rádiových otisků. Charakteristika MAC address bude vracet MAC adresu rádiového otisku podle nastaveného indexu charakteristiky Set index. Charakteristika RSSI bude vracet RSSI rádiového otisku podle nastaveného indexu charakteristiky Set index. Charakteristika Age of record bude vracet čas pořízení rádiového otisku během skenování v milisekundách, tj. skenování probíhá od časového okamžiku A po dobu trvání skenu do časového okamžiku B. Pokud by skenování probíhalo 5000 ms a rádiový otisk by byl pořízen v čase 1000 ms, hodnota charakteristiky Age of record bude právě 1000. Hodnota této charakteristiky je též závislá na nastaveném indexu charakteristiky Set index. Charakteristika Flag bude vracet příznak v případě, že bylo nalezeno více zařízení, která se již nevešla do paměti. Poslední charakteristika Age of scan bude vracet čas od zahájení skenování.

Nyní je zřejmé, že by bylo vhodné tyto dvě služby seskupit do jednoho profilu. Profil se bude jmenovat Beacons Scanner Profile. Celé seskupení profilu, služeb a charakteristik je pro větší přehlednost znázorněno v tabulce 1.

Tabulka 1 Přehled služeb a charakteristik

	Služba	Charakteristika	Čtení/ Zápis	Rozsah hodnot	Příklad hodnoty[4]	Význam hodnoty	Popis
Beacons Scanner Profile	Beacons discovery	Start scan	R/W	0 - 0xFFFF	8813	5000 ms	Zahájení skenu na dobu, podle přijaté hodnoty
	Beacons list	Set index	W	0 - (total count – 1)	0500	Šestý záznam	Nastavení indexu záznamu
		Total count	R	0 - 0xFFFF	F401	500 záznamů	Celkový počet pořízených rádiových otisků
		MAC address	R	0 - 0xFFFFFFFFFFFF	44C11031CCCD	CD:CC:31:10:C1:44	MAC adresa iBeaconu
		RSSI	R	0 - 0xFF	2A	-42	RSSI iBeaconu
		Age of record	R	0 - 0xFFFF	5203	850 ms	Čas, ve kterém byl rádiový otisk pořízen, během skenu
		Flag	R	0 - 0x01	01	Příznak	Pokud Sensor Tag objevil během skenování více zařízení, která se již nevešla do paměti
		Age of scan	R	0 - 0xFFFF	B888	35000 ms	Doba od zahájení skenu

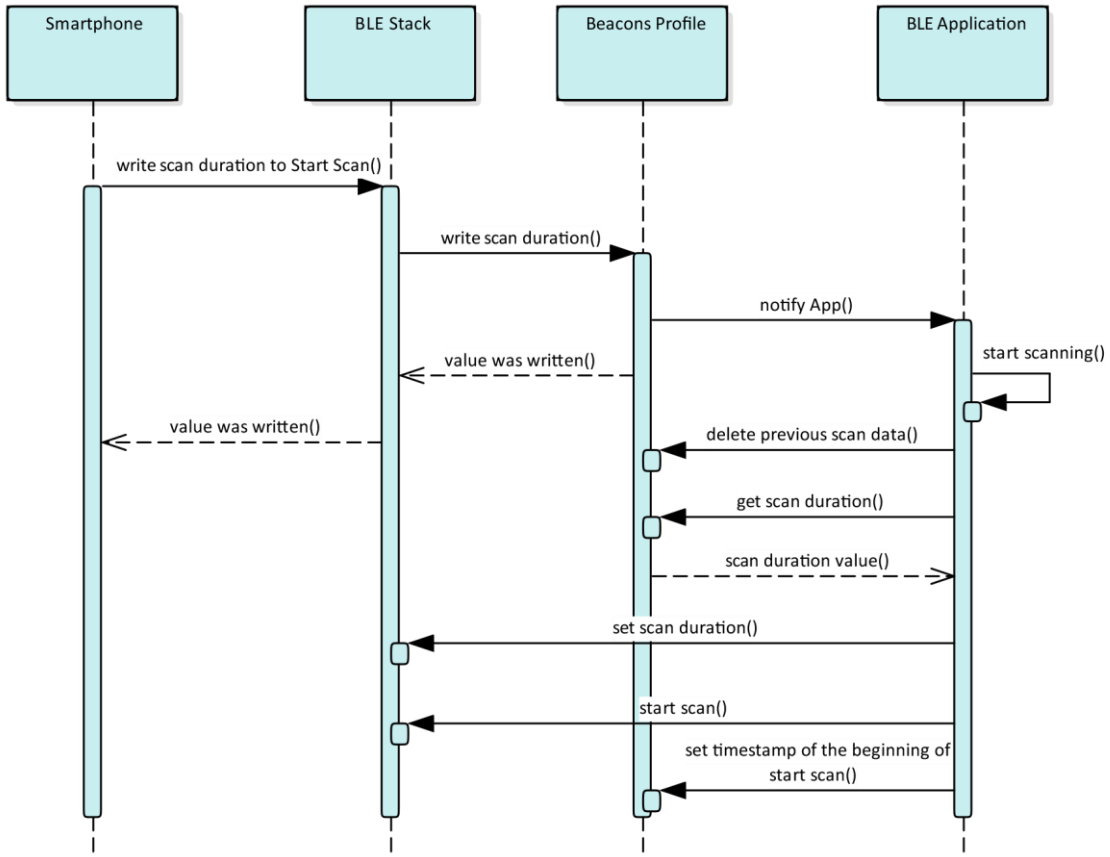
Zdroj vlastní tvorba

[4] hodnoty jsou zapsány v Little-Endian

5.2.2 Zahájení skenování

Komunikace mezi smartphonem a Sensor Tagem probíhá prostřednictvím BLE stacku od Texas Instruments. Pokud chce smartphone zahájit skenování okolních iBeaconů pomocí Sensor Tagu, musí poslat skrz Bluetooth hodnotu určující trvání skenu. BLE stack hodnotu přijme a zapíše ji do charakteristiky Start scan nacházející se v Beacons Scanner Profilu. Po úspěšném zapsání hodnoty do Start scan charakteristiky BLE stack informuje smartphone o této skutečnosti. Beacons Scanner Profile po zapsání hodnoty určující trvání skenu informuje aplikaci o zahájení skenu. V aplikaci dojde po zpracování oznámení k zavolání metody starající se o zahájení skenu. V této metodě se uskuteční několik následujících kroků.

V prvním kroku se vynulují veškeré potřebné čítače, které udržují informace o rádiových otiscích získaných z předešlého skenování. Dále aplikace požádá Beacons Scanner Profile o hodnotu charakteristiky Start scan, ve které je uložena hodnota určující trvání skenování. Po obdržení hodnoty aplikace požádá BLE stack o nastavení hodnoty scan duration na obdrženou hodnotu. Tímto krokem dojde k nastavení délky trvání skenování okolních iBeaconů. V následujícím kroku aplikace zavolá metodu pro zahájení skenování a BLE stack zahájí skenování. V posledním kroku aplikace nastaví timestamp začátku skenu do charakteristiky Age of scan v Beacons Scanner Profilu. Celý průběh komunikace je znázorněn v sekvenčním diagramu na Obr. 9.

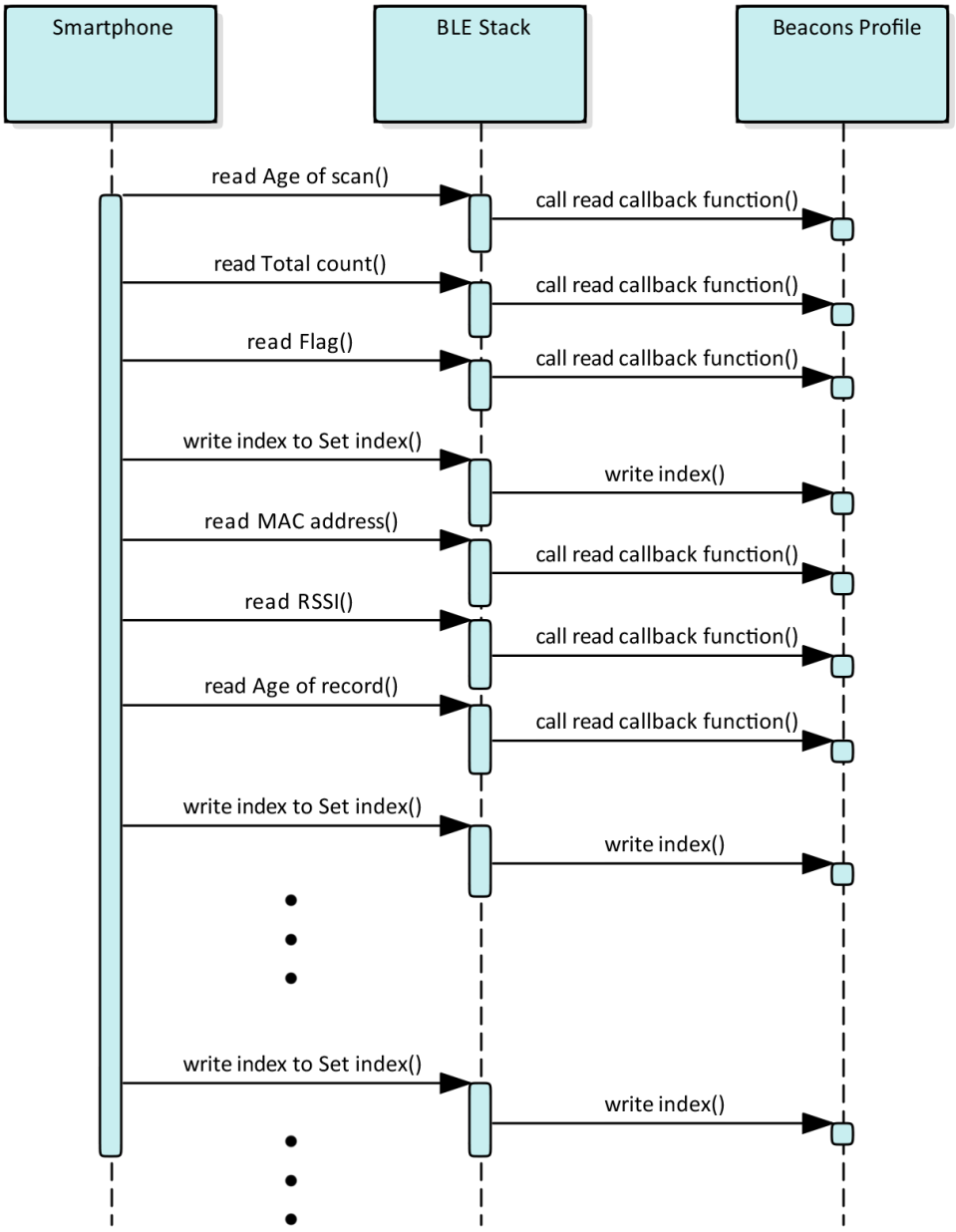


Obr. 9 Sekvenční diagram znázorňující komunikaci mezi entitami před zahájením skenování. Zdroj vlastní tvorba

5.2.3 Čtení dat z profilu

Po dokončení skenování začne smartphone postupně číst data získaná během skenování ze Sensor Tagu. Čtení dat probíhá v následujících krocích. Nejprve jsou vyčtena data charakterizující právě dokončený sken. Jsou to hodnoty charakteristik Age of scan, Total count a Flag. Hodnota charakteristiky Age of scan udává stáří skenu, jak již bylo zmíněno výše. Je tedy žádoucí číst tuto hodnotu jako první. Hned poté je přečtena hodnota charakteristiky Total count. Na základě této hodnoty jsou poté čteny rádiové otisky. V první řadě smartphone zapíše index požadovaného rádiového otisku do charakteristiky Set index. Poté následuje čtení 3 charakteristik charakterizujících rádiový otisk. Jedná se tedy o tyto charakteristiky: MAC address, RSSI a Age of record. Jakmile jsou data z těchto charakteristik přečtená, smartphone zapíše index následujícího rádiového otisku. Následuje opět čtení již zmíněných 3 charakteristik. Tyto kroky se opakují, dokud

nejsou smartphonem přečteny všechny rádiové otisky získané během skenování, tedy dokud jde zapisovat index následujícího rádiového otisku. Hodnota indexu je vždy v intervalu od 0 do hodnoty charakteristiky *Total count – 1*. Průběh této komunikace je znázorněn v sekvenčním diagramu na Obr. 10.



Obr. 10 Sekvenční diagram znázorňující čtení dat získaných během skenování smartphonem ze Sensor Tagu. Zdroj vlastní tvorba

6 Implementace

Pro implementaci bylo použito IDE Code Composer Studio ve verzi 7.2.0.00013. Dále je nezbytné mít nainstalovaný BLE stack ve verzi 2.2.1. Je vřele doporučováno, aby instalace BLE stacku byla umístěna ve výchozí adresářové struktuře, tedy ve složce ti v kořenovém adresáři disku C. Jako základ aplikace byl využit ukázkový projekt od Texas Instruments simple_peripheral_observer dostupný z githubu[5] Texas Instruments. Aby bylo možné naimportovat výslednou aplikaci vytvořenou v rámci této bakalářské práce, je nezbytné stáhnout tyto příklady. Po stažení příkladů je nutné rozbalit kořenový adresář ble_examples-ble_examples-2.2 z archivu do adresáře BLE stacku, tedy do C:\ti\simplelink\ble_sdk_2_02_01_18. Tento ukázkový projekt byl zvolen z důvodu potřeby skenovat okolní iBeacon zařízení a poté informace o naskenovaných zařízeních poskytnout jinému bluetooth zařízení. Jelikož nebylo potřeba přímo navázat spojení mezi Sensor Tagem a iBeacon zařízeními, byla zvolena Observer role, která umožňuje skenování okolních zařízení a přijímání advertise paketů. Tato role je o něco méně náročná na hardware oproti Central roli, která umožňuje již zmíněné navázání spojení s jiným Peripheral zařízením. Samotná implementace je realizována v programovacím jazyce C.

6.1 Start programu – funkce main

Veškerá inicializace začíná ve funkci main, která je součástí souboru main.c. V této funkci je zaregistrován aplikační callback, který reaguje na chyby vzniklé za běhu v BLE stacku. Tento callback reaguje dle výchozího chování, a to tím způsobem, že na displej Sensor Tagu vypíše vzniklou chybu. Potom následuje inicializace pinů, zakázaní přerušení a inicializace ovladačů podle zvolené vývojové desky. Dále následuje nastavení power saving v případě, že je definován patřičný předdefinovaný symbol pro preprocesor. Hned poté je nastaveno, aby obsah 8 kB cache zůstal i v pohotovostním režimu. Toto nastavení je důležité, neboť 8 kB cache je využívána jako GPRAM (General Purpose Random Access Memory) v důsledku nedostatku paměti 20 kB SRAM. V dalším kroku dojde k inicializaci

[5] https://github.com/ti-simplelink/ble_examples/tree/ble_examples-2.2

samotného ICall modulu a vytvoření tzv. tasků pro ICall, které mají nejvyšší prioritu. Dále následuje vytvoření GAPRole tasku tvořící API konkrétní bluetooth role pro samotnou aplikaci. Posledním vytvářeným taskem se stává task samotné aplikace. Tento aplikační task má nejnižší prioritu. V poslední řadě dojde k povolení přerušení a k zahájení činnosti plánovače SYS/BIOS kernelu zavoláním funkce BIOS_start [13]. Zdrojový kód právě popsané funkce main je uveden v ukázce kódu č. 1.

```
int main()
{
  /* Register Application callback to trap asserts raised in the Stack */
  RegisterAssertCback(AssertHandler);

  PIN_init(BoardGpioInitTable);

  #ifndef POWER_SAVING
  /* Set constraints for Standby, powerdown and idle mode */
  Power_setConstraint(PowerCC26XX_SB_DISALLOW);
  Power_setConstraint(PowerCC26XX_IDLE_PD_DISALLOW);
  #endif // POWER_SAVING

  // retain cache during standby
  Power_setConstraint(PowerCC26XX_SB_VIMS_CACHE_RETAIN);
  Power_setConstraint(PowerCC26XX_NEED_FLASH_IN_IDLE);

  /* Initialize ICall module */
  ICall_init();

  /* Start tasks of external images - Priority 5 */
  ICall_createRemoteTasks();

  /* Kick off profile - Priority 3 */
  GAPRole_createTask();

  SimpleBLEPeripheral_createTask();

  /* enable interrupts and start SYS/BIOS */
  BIOS_start();

  return 0;
}
```

Ukázka kódu č. 1

6.2 Aplikační část

V této podkapitole jsou popsány stěžejní části aplikace. Veškeré popsané funkce v této podkapitole se nacházejí v souboru simple_peripheral_observer.c.

6.2.1 Aplikační task

Task aplikace je vytvořen po zavolání funkce SimpleBLEPeripheral_createTask pomocí struktury Task_Struct. V této funkci se nastaví parametry vytvářeného tasku pomocí struktury Task_Params_init. Reference na tuto strukuru je poté předána funkci Task_construct, která je zodpovědná za vytvoření tasku, jako jeden z parametrů. Při vytváření tasku je vždy nutné definovat tyto tři parametry: reference na stack tasku, velikost tohoto stacku v bajtech a priorita tasku, přičemž hodnota priority pro aplikační task je zpravidla nejnižší a jedná se o hodnotu 1. Texas Instruments nedoporučuje nastavovat prioritu aplikačního tasku na hodnotu větší nebo rovnou hodnotě priority BLE stack tasku nebo s ním souvisejícím taskům např. GAPRole task. Tasku je možno nastavit další různé parametry. Task je vytvořen po zavolání funkce Task_construct, které se předají tyto čtyři parametry: reference na strukturu Task_Struct, reference na funkci SimpleBLEPeripheral_taskFxn, reference na strukturu Task_Params_init, kde jsou parametry tasku a reference na strukturu Error_Block. Struktura Error_Block zde není využívána, a proto je místo reference na tuto strukturu předána funkci Task_construct hodnota NULL [13]. Výše popsaný zdrojový kód je uveden níže v ukázce kódu č. 2.

```
void SimpleBLEPeripheral_createTask(void)
{
  Task_Params taskParams;

  // Configure task
  Task_Params_init(&taskParams);
  taskParams.stack = sbpTaskStack;
  taskParams.stackSize = SBP_TASK_STACK_SIZE;
  taskParams.priority = SBP_TASK_PRIORITY;

  Task_construct(&sbpTask, SimpleBLEPeripheral_taskFxn, &taskParams,
                 NULL);
}
```

Ukázka kódu č. 2

Funkce SimpleBLEPeripheral_taskFxn je vykonávána v době, kdy aplikační task dostane prostředky od TI-RTOS. Tato funkce je tedy velice významná, neboť umožňuje spustit kód aplikace, kterou programátor napíše. V této funkci je nejprve zavolána funkce SimpleBLEPeripheral_init, která je popsána dále v textu.

Po vykonání funkce SimpleBLEPeripheral_init následuje nekonečný cyklus. V tomto cyklu se zpracovávají zprávy přijaté od okolních tasků. Tyto zprávy jsou uložené ve frontě a zpracovávají se tedy v pořadí, ve kterém přišly. Dále v tomto nekonečném cyklu může být zpracovávána periodická úloha. Tento nekonečný cyklus není vykonáván neustále, jelikož TI-RTOS tento task po určité době uspí z důvodu úspory energie. Pokud tedy entita pošle aplikaci zprávu, musí také poslat signál semaforu, aby byl aplikační task znovu probuzen. Náhled funkce SimpleBLEPeripheral_taskFxn se nachází v ukázce kódu č. 3.

```
static void SimpleBLEPeripheral_taskFxn(UArg a0, UArg a1)
{
  // Initialize application
  SimpleBLEPeripheral_init();

  // Application main loop
  for (;;)
  {
      .
      . // Processing of messages
      .
  }
```

Ukázka kódu č. 3

6.2.2 Inicializační funkce aplikace

Funkce SimpleBLEPeripheral_init je stěžejní, protože zde proběhne celá inicializace aplikace. V první řadě pro aplikace využívající bluetooth je nezbytné zaregistrovat vlákno aplikace jako ICall dispečer pomocí funkce ICall_registerApp. Této funkci se předají dva parametry: reference na frontu zpráv a reference na semafor. Kdyby nebyla aplikace zaregistrována jako ICall dispečer, nebyla by schopna komunikovat s BLE stackem skrze ICall. V následujícím kroku se vytvoří fronta pro zprávy, pomocí funkce Util_constructQueue, které jsou přijímány od profilů, případně dalších okolních entit, které chtějí zaslat zprávu aplikaci. Dále se otevře kanál pro komunikaci s displejem skrze funkci Display_open, která vrátí referenci na strukturu Display_Handle. Pomocí této reference je poté již možné zapisovat na displej. Část zdrojového kódu inicializační funkce je uvedena v ukázce kódu č. 4.

```
static void SimpleBLEPeripheral_init(void)
{
  // ******************************************************************
  // N0 STACK API CALLS CAN OCCUR BEFORE THIS CALL TO ICall_registerApp
  // ******************************************************************
  // Register the current thread as an ICall dispatcher application
  // so that the application can send and receive messages.
  ICall_registerApp(&selfEntity, &sem);

  // Create an RTOS queue for message from profile to be sent to app.
  appMsgQueue = Util_constructQueue(&appMsg);

  dispHandle = Display_open(Display_Type_LCD, NULL);
```

Ukázka kódu č. 4

Nyní následuje inicializační blok pro Observer roli. Zde se nastaví maximální počet reportů, které BLE stack podá aplikaci během skenování. Dále se nastaví výchozí délka skenování, sken interval, sken window a zda má BLE stack filtrovat reporty zachycených advertise paketů. Jinými slovy, jestli má BLE stack aplikaci informovat o každém zachyceném advertise paketu od stejného zařízení, které jej vyslalo. Tato možnost je nastavena na hodnotu FALSE, protože snahou je vytvořit zařízení, které zachytí co nejvíce advertise paketů od okolních iBeacon zařízení. Inicializační blok observer role je znázorněn v ukázce kódu č. 5.

```
  //Setup GAP Observer params
  {
    uint8_t scanRes = DEFAULT_MAX_SCAN_RES;

    GAPRole_SetParameter(GAPROLE_MAX_SCAN_RES, sizeof(uint8_t),
                               &scanRes);
    // Set the GAP Characteristics
    //how long to scan (in scan state)
    GAP_SetParamValue(TGAP_GEN_DISC_SCAN, DEFAULT_SCAN_DURATION);
    GAP_SetParamValue(TGAP_LIM_DISC_SCAN, DEFAULT_SCAN_DURATION);

    //Set scan interval
    //period for one scan channel
    GAP_SetParamValue(TGAP_GEN_DISC_SCAN_INT,
                      (DEFAULT_SCAN_INTERVAL)/(0.625));

    //Set scan window
    //active scanning time within scan interval
    GAP_SetParamValue(TGAP_GEN_DISC_SCAN_WIND,
                      (DEFAULT_SCAN_WINDOW)/(0.625));

    GAP_SetParamValue(TGAP_FILTER_ADV_REPORTS, FALSE);
  }
```

Ukázka kódu č. 5

Dále následuje inicializační blok pro Peripheral roli. Prvním nastaveným parametrem je GAPROLE_ADVERT_ENABLED. Tento parametr nastavuje, zda má Sensor Tag posílat advertise pakety, respektive zda je Sensor Tag viditelný pro ostatní zařízení ihned po jeho zapnutí. Parametr je nastaven na hodnotu TRUE. Druhým nastaveným parametrem je GAPROLE_ADVERT_OFF_TIME, kdy tento parametr má význam v případě, že by Sensor Tag byl nastavený v limited discovery módu. Nastavení tohoto parametru na hodnotu 0 zapříčiní, že po uplynutí doby 30,72 sekund již nebude Sensor Tag dále zjistitelný pro ostatní bluetooth zařízení, dokud se opět nenastaví parametr GAPROLE_ADVERT_ENABLED na hodnotu TRUE. Následuje nastavení parametru GAPROLE_SCAN_RSP_DATA, pomocí kterého se nastaví obsah scan response paketu. Posledním nastaveným parametrem je GAPROLE_ADVERT_DATA. Skrze tento parametr se nastaví obsah advertise paketu vysílaný Sensor Tagem. Nastavení těchto parametrů je znázorněno v ukázce kódu č. 6.

```
// Setup the GAP Peripheral Role Profile
{
  // For all hardware platforms, device starts advertising upon
  // initialization
  uint8_t initialAdvertEnable = TRUE;

  // By setting this to zero, the device will go into the waiting state
  // after being discoverable for 30.72 second, and will not being
  // advertising again until the enabler is set back to TRUE
  uint16_t advertOffTime = 0;

  // Set the GAP Role Parameters
  GAPRole_SetParameter(GAPROLE_ADVERT_ENABLED, sizeof(uint8_t),
                       &initialAdvertEnable);
  GAPRole_SetParameter(GAPROLE_ADVERT_OFF_TIME, sizeof(uint16_t),
                       &advertOffTime);

  GAPRole_SetParameter(GAPROLE_SCAN_RSP_DATA, sizeof(scanRspData),
                       scanRspData);
  GAPRole_SetParameter(GAPROLE_ADVERT_DATA, sizeof(advertData),
                        advertData);
}
```

Ukázka kódu č. 6

V dalším kroku se nastaví jméno zařízení v charakteristice vrstvy GAP. Pomocí této charakteristiky si ostatní zařízení mohou načíst jméno Sensor Tagu a

případně ho zobrazit. Jméno Sensor Tagu je nastaveno na hodnotu CC2650 Sensor Tag. Následuje nastavení advertise intervalu. Tento interval udává, jak často má Sensor Tag vysílat advertise pakety. Nastavení advertise intervalu proběhne pro oba discovery módy na stejnou hodnotu, tedy pro limited i general discovery mód. Advertise interval je nastaven na hodnotu 1000 ms. V ukázce kódu č. 7 je uvedeno nastavení zmíněných parametrů.

```
// Set the GAP Characteristics
GGS_SetParameter(GGS_DEVICE_NAME_ATT, ARRAY_SIZE(attDeviceName),
                 attDeviceName);

// Set advertising interval
{
  uint16_t advInt = DEFAULT_ADVERTISING_INTERVAL;

  GAP_SetParamValue(TGAP_LIM_DISC_ADV_INT_MIN, advInt);
  GAP_SetParamValue(TGAP_LIM_DISC_ADV_INT_MAX, advInt);
  GAP_SetParamValue(TGAP_GEN_DISC_ADV_INT_MIN, advInt);
  GAP_SetParamValue(TGAP_GEN_DISC_ADV_INT_MAX, advInt);
}
```

Ukázka kódu č. 7

Dále je zde nastíněná ukázka GAP bond manageru od Texas Instruments. Jedná se o druh zabezpečení, kdy Peripheral zařízení může nejprve vyžadovat provedení tzv. párování před zahájením komunikace. Spárování proběhne pouze v případě, kdy si Central i Peripheral zařízení vymění párovací klíč. Existuje několik způsobů, jak se dá toto párování nakonfigurovat. Jednou z možností je takové párování, kdy Peripheral zařízení vyžaduje nejprve provést spárování s Central zařízením. Toho může být dobře využito, pokud se Peripheral zařízení nachází na veřejně dostupném místě a zároveň je nutné, aby bylo kdykoliv přístupné. V této ukázce je párování použito pouze v případě, kdy Central zařízení pošle požadavek na párování Sensor Tagu. GAP bond manager může být využit, pokud je definován patřičný předdefinovaný symbol pro preprocesor. Pro účely této práce není GAP bond manager využitý a slouží zde pouze jako příklad uvedený v ukázce kódu č. 8.

```
#ifdef PLUS_BOND_MANAGER
// Setup the GAP Bond Manager
{
  uint32_t passkey = 0; // passkey "000000"
  uint8_t pairMode = GAPBOND_PAIRING_MODE_WAIT_FOR_REQ;
  uint8_t mitm = TRUE;
  uint8_t ioCap = GAPBOND_IO_CAP_DISPLAY_ONLY;
  uint8_t bonding = TRUE;

  GAPBondMgr_SetParameter(GAPBOND_DEFAULT_PASSCODE, sizeof(uint32_t),
                          &passkey);
  GAPBondMgr_SetParameter(GAPBOND_PAIRING_MODE, sizeof(uint8_t),
                          &pairMode);
  GAPBondMgr_SetParameter(GAPBOND_MITM_PROTECTION, sizeof(uint8_t),
                          &mitm);
  GAPBondMgr_SetParameter(GAPBOND_IO_CAPABILITIES, sizeof(uint8_t),
                          &ioCap);
  GAPBondMgr_SetParameter(GAPBOND_BONDING_ENABLED, sizeof(uint8_t),
                          &bonding);
}
#endif
```

Ukázka kódu č. 8

V následujících krocích proběhne inicializace GATT atributů. V první řadě musí být inicializována GATT tabulka vrstvy GAP. Toho je docíleno zavoláním funkce GGS_AddService. Poté musí dojít k inicializaci samotné vrstvy GATT zavoláním funkce GATTServApp_AddService. Po inicializaci GATT vrstvy je u této vrstvy možné zaregistrovat libovolné vlastní služby a charakteristiky. Pro účely této práce je vytvořen Beacons Scanner Profile, který obsahuje dvě vlastní služby. Služby Beacons Scanner Profilu jsou inicializovány v GATT tabulce zavoláním funkce BeaconsScannerProfile_AddService. Podrobnější popis této funkce je popsán v kapitole **6.3.2 Add service funkce**. Dále je funkcí BeaconsScannerProfile_RegisterAppCBs zaregistrován aplikační callback v Beacons Scanner Profilu, kdy v parametru funkce je na tento callback předána reference. Tento callback slouží Beacons Scanner Profilu pro notifikování aplikace v případě, kdy je v charakteristice Start scan zapsána určitá hodnota. Detailnější popis využití aplikačního callbacku pro Beacons Scanner Profile je popsán v kapitole **6.3.3 Funkce registrující aplikační callback funkci**. Pořadí volání funkcí inicializující GATT vrstvu je uvedeno v ukázce kódu č. 9.

```
 // Initialize GATT attributes
GGS_AddService(GATT_ALL_SERVICES);           // GAP
GATTServApp_AddService(GATT_ALL_SERVICES);   // GATT attributes
BeaconsScannerProfile_AddService();

BeaconsScannerProfile_RegisterAppCBs(
                              &SimpleBLEPeripheral_beaconsProfileCBs);
```

Ukázka kódu č. 9

Zavoláním funkce GAPRole_StartDevice dojde k nastartování bluetooth rolí a celkové bluetooth aktivitě. V případě Peripheral role začne Sensor Tag vysílat advertise pakety a stane se tak zjistitelným pro ostatní zařízení. Této funkci je předána reference na callback SimpleBLEPeripheral_gapRoleCBs, který slouží pro zachytávání událostí vzniklých během bluetooth aktivity. Podrobnější popis tohoto callbacku je v kapitole **6.2.3 Observer callback funkce**. V neposlední řadě se zavoláním funkce GAPBondMgr_Register spustí GAP bond manager, pokud je definován patřičný předdefinovaný symbol pro preprocesor. I zde se opět předává callback funkce. Jak již bylo zmíněno výše, pro účely této práce není GAP bond manager využitý a slouží zde pouze jako ukázka uvedená níže v kódu č. 10.

```
// Start the Device
VOID GAPRole_StartDevice(&SimpleBLEPeripheral_gapRoleCBs);

#ifdef PLUS_BOND_MANAGER
// Start Bond Manager
VOID GAPBondMgr_Register(&simpleBLEPeripheral_BondMgrCBs);
#endif
```

Ukázka kódu č. 10

V poslední řadě se aplikace zaregistruje zavoláním funkce GAP_RegisterForMsgs pro získání více událostí vzniklých v GAP rolích. Funkci se jako parametr předá ID aplikačního tasku. Dále se aplikace zaregistruje u GATT vrstvy pro získání událostí a ATT odpovědí, které nebyly odeslány Central zařízení v důsledku nedostatku paměti HCI (Host Controller Interface) bufferů. Tyto ATT odpovědi se poté aplikace pokusí znovu odeslat [13]. Registrace proběhne při zavolání funkce GATT_RegisterForMsgs, kdy parametrem této funkce se předá ID aplikačního tasku. Úplně nakonec se na první řádek displeje Sensor Tagu vypíše

název zařízení, tedy CC2650-Indoor. Ukázka kódu č. 11 zobrazuje konec inicializační funkce.

```
  // Register with GAP for HCI/Host messages
  GAP_RegisterForMsgs(selfEntity);

  // Register for GATT local events and ATT Responses pending for
  transmission
  GATT_RegisterForMsgs(selfEntity);

  Display_print0(dispHandle, 0, 1, "CC2650-Indoor");
}
```

Ukázka kódu č. 11

6.2.3 Observer callback funkce

V tomto callbacku probíhá zpracovávání událostí, které zaslal BLE stack aplikaci. Událost je předána pomocí reference na strukturu gapPeripheralObserverRoleEvent_t, ve které jsou veškeré informace o nastalé události. Pro tuto práci jsou stěžejní právě tyto dvě události: GAP_DEVICE_INFO_EVENT a GAP_DEVICE_DISCOVERY_EVENT.

Událost GAP_DEVICE_INFO_EVENT pošle BLE stack v případě, kdy zachytí advertise paket od okolního zařízení. Nejprve se zvýší obsah proměnné rspCount o jedničku a poté se vypíše na displej Sensor Tagu hodnota této proměnné, která počítá počet přijatých advertise paketů. Poté se zavolá funkce BeaconsScannerProfile_AddBeaconRecord, která má za úkol uložit přijaté informace o zařízení, respektive rádiový otisk. Této funkci se předají následující tři parametry: MAC adresa zařízení, hodnota RSSI a časové razítko. Podrobný popis této funkce je v kapitole **6.3.7 Ukládání rádiových otisků**. Nakonec se pomocí funkce ICall_free uvolní paměť, která byla obsazena touto událostí.

Druhá událost GAP_DEVICE_DISCOVERY_EVENT je zaslána BLE stackem jakmile skenování skončí, zpravidla po uplynutí doby skenování. V první řadě se vypíše na displej Sensor Tagu informace, že skenování skončilo. Dále se nastaví hodnota charakteristiky Start scan na 0. To je provedeno zavoláním funkce BeaconsScannerProfile_SetParameter, kdy této funkci se předá ukazatel na hodnotu 0. Více informací o této funkci je v kapitole **6.3.4 SET funkce**. Následuje

nastavení hodnoty proměnné rspCount na 0, aby při dalším skenování bylo možné spočítat počet přijatých advertise paketů. V poslední řadě dojde opět k uvolnění paměti pomocí funkce ICall_free. Celá funkce tvořící Observer callback je uvedena v ukázce kódu č. 12.

```
static void SimpleBLEPeripheralObserver_processRoleEvent(
gapPeripheralObserverRoleEvent_t *pEvent)
{
  switch (pEvent->gap.opcode)
  {

    case GAP_DEVICE_INFO_EVENT:
      {
        Display_print1(dispHandle, 6, 0, "RspCount: %d", ++rspCount);
        BeaconsScannerProfile_AddBeaconRecord(pEvent->deviceInfo.addr,
                                              pEvent->deviceInfo.rssi,
                                              Timestamp_get32());

        ICall_free(pEvent->deviceInfo.pEvtData);
        ICall_free(pEvent);
      }
      break;

    case GAP_DEVICE_DISCOVERY_EVENT:
      {
        Display_print0(dispHandle, 5, 0, "Scanning Off");
        uint16 stopScan = 0;
        BeaconsScannerProfile_SetParameter(BEACONS_DISCO_SCAN,
                                           sizeof(uint16),
                                           &stopScan);

        rspCount = 0;

        ICall_free(pEvent->discCmpl.pDevList);
        ICall_free(pEvent);

      }
      break;
  }
}
```

Ukázka kódu č. 12

6.2.4 Funkce zahajující skenování

Z názvu funkce SimpleBLEPeripheral_startDiscovery je zřejmé, že daná funkce je zodpovědná za zahájení skenování. Tato funkce je zavolána aplikací ve chvíli, kdy je zaslána aplikaci notifikace od Beacons Scanner Profilu určující zahájení skenování. V prvním kroku se vynulují čítače určující počet platných

záznamů v paměti, příznak určující nalezení více zařízení a nastavení indexu aktuálního záznamu na hodnotu 0. Toto je docíleno zavoláním funkce BeaconsScannerProfile_ResetCounters. V následujícím kroku dojde k nastavení hodnoty určující dobu skenování. Tato hodnota se nastaví na takovou hodnotu, která byla po přijetí zapsána do charakteristiky Start scan. Nejprve je tedy nutné získat adresu paměti, na které se nachází hodnota charakteristiky Start scan. Toho je docíleno zavoláním funkce BeaconsScannerProfile_GetParameter, kde v parametru této funkce je předáno identifikační číslo dané charakteristiky. Jakmile je získána adresa paměti, na které se nachází hodnota určující dobu skenování, je hodnota na této adrese předána funkci GAP_SetParamValue. Tímto se nastaví doba skenování pro oba skenovací módy, tedy general discovery mode i limited discovery mode. V následujícím kroku zavoláním funkce GAPObserverRole_StartDiscovery se pokusí BLE stack zahájit skenování. Této funkci se předávají tři parametry: skenovací mód, zdali má být sken aktivní a tzv. white list. Skenovací mód může být general, limited nebo all. Zde je hodnota all, tudíž se použijí oba zmíněné skenovací módy. Aktivní sken znamená, že Sensor Tag během skenování okolních zařízení vysílá scan request pakety a ostatní peripheral zařízení na tyto pakety odpovídají pomocí scan response paketů. Tato hodnota je nastavená na hodnotu TRUE, je tedy využíván aktivní sken. Aktivní sken je využitý, protože bylo zjištěno, že iBeacon zařízení z důvodu úspory energie vysílají advertise pakety v dlouhých intervalech. Jakmile iBeacon příjme scan request paket, odpoví na tento paket scan response paketem a začne vysílat advertise pakety v intenzitě, kterou má nastavenou. Posledním parametrem je white list. Pomocí white listu je možné definovat, o kterých zařízeních má BLE stack informovat aplikaci. Jedná se tedy o seznam MAC adres, které je možné předat BLE stacku. Pro účely této práce se tato možnost nehodí a není tedy využívána, hodnota je nastavena na FALSE. Pokud proběhne vše v pořádku, BLE stack zahájí skenování a funkce GAPObserverRole_StartDiscovery vrátí hodnotu SUCCESS. V opačném případě vrátí hodnotu chybového kódu. Tato vrácená hodnota je uložena do proměnné status, podle které proběhne následující rozhodování. Pokud je v proměnné status hodnota SUCCESS, následuje získání časového razítka funkcí Timestamp_get32. Poté je adresa paměti na toto časové razítko předána funkci

BeaconsScannerProfile_SetParameter. V této funkci dojde k uložení časového razítka do charakteristiky Age of scan. Nakonec se na displej Sensor Tagu vypíše informace, že Sensor Tag skenuje a přepíše se hodnota rspCount, která zůstala na displeji z předchozího skenování. Na displeji se dále vypíše doba trvání skenu. V opačném případě, kdy hodnota proměnné status není SUCCESS, se na displej Sensor Tagu vypíše skutečnost, že se nezdařilo zahájit skenování a vypíše se číslo chyby, které je uložené v proměnné status. V ukázce kódu č. 13 je možné vidět deklaraci funkce SimpleBLEPeripheral_startDiscovery.

```
void SimpleBLEPeripheral_startDiscovery(void)
{
    BeaconsScannerProfile_ResetCounters();

    uint16 * scanDuration = (uint16 *)
BeaconsScannerProfile_GetParameter(BEACONS_DISCO_SCAN);

    GAP_SetParamValue(TGAP_GEN_DISC_SCAN, *scanDuration);
    GAP_SetParamValue(TGAP_LIM_DISC_SCAN, *scanDuration);

    bStatus_t status = GAPObserverRole_StartDiscovery(
                                DEFAULT_DISCOVERY_MODE,
                                DEFAULT_DISCOVERY_ACTIVE_SCAN,
                                DEFAULT_DISCOVERY_WHITE_LIST);

    if(status == SUCCESS)
    {
        uint32_t startTime = Timestamp_get32();
        BeaconsScannerProfile_SetParameter(
                                BEACONS_LIST_AGE_OF_SCAN,
                                sizeof(startTime),
                                &startTime);

        Display_print1(dispHandle, 6, 0, "RspCount: %d", rspCount);
        Display_print0(dispHandle, 5, 0, "Scanning On");
        Display_print1(dispHandle, 7, 0, "Duration: %d", *scanDuration);
    }
    else
    {
        Display_print1(dispHandle, 5, 0, "Scanning Fail:%d", status);
    }
}
```

Ukázka kódu č. 13

6.3 Beacons Scanner Profile

Jedná se o profil obsahující dvě služby, jak již bylo zmíněno výše v textu. Tento profil je stěžejní, neboť zprostředkovává komunikaci mezi Sensor Tagem a jiným Central zařízením. V této kapitole je tedy podrobně popsán.

6.3.1 Deklarace atributů a sestavení ATT tabulky

V této podkapitole je popsáno, jak jsou služby a charakteristiky definovány skrze vrstvu ATT. Nejprve je popsána deklarace jednotlivých služeb a charakteristik a poté sestavení ATT tabulky.

Nejdříve jsou tedy deklarovány všechny UUID jednotlivých charakteristik a služeb. V textu je UUID zapisováno v šestnáctkové soustavě. Vlastní služby a charakteristiky mají 16 bajtové UUID, jak již bylo zmíněno výše v textu. Deklarovány jsou pouze dva z šestnácti bajtů. Pro vygenerování celého 16 bajtového UUID je využito makro TI_BASE_UUID_128 definované společností Texas Instruments. Toto makro vrátí výsledné UUID ve tvaru, ze kterého se inicializuje pole bajtů dlouhé šestnáct prvků, tj. toto pole je dlouhé šestnáct bajtů. Výsledné UUID je ve tvaru F000XXXX-0451-4000-B000-000000000000, kde XXXX jsou dva bajty definované unikátně pro každou službu a charakteristiku. UUID pro jednotlivé služby bylo zvoleno ve tvaru X000, kde X určuje jednotlivou službu a začíná od symbolu A. Jednotlivé charakteristiky potom mají v dané službě UUID ve tvaru XY00, kde X určuje danou službu a Y konkrétní charakteristiku uvnitř služby, přičemž Y začíná od hodnoty 1. Služba Beacons discovery má tedy UUID A000 a charakteristika Start scan má potom UUID A100. Deklarace UUID všech služeb a charakteristik je uvedena v ukázce kódu č. 14.

```
#define BEACONS_DISCO_SERVICE_UUID          0xA000
#define BEACONS_DISCO_SCAN_UUID             0xA100

#define BEACONS_LIST_SERVICE_UUID           0xB000
#define BEACONS_LIST_GET_RECORD_UUID        0xB100
#define BEACONS_LIST_TOTAL_COUNT_UUID       0xB200
#define BEACONS_LIST_MAC_ADDR_UUID          0xB300
#define BEACONS_LIST_RSSI_UUID              0xB400
#define BEACONS_LIST_AGE_UUID               0xB500
#define BEACONS_LIST_FLAG_OF_MAC_UUID       0xB600
#define BEACONS_LIST_AGE_OF_SCAN_UUID       0xB700
```

Ukázka kódu č. 14

Po nadefinování UUID pro služby i charakteristiky následuje nadefinování dalších nezbytných komponent, ze kterých se poté sestaví ATT tabulka. V následující části textu je již popsána deklarace komponent charakteristiky Start scan a sestavení ATT tabulky pro službu Beacons discovery. Nejprve je potřeba definovat pomocí struktury gattAttrType_t UUID služby, kdy tato struktura obsahuje dvě položky: délku UUID a referenci na pole obsahující UUID. Takto vyjádřené UUID pomocí struktury gattAttrType_t je uloženo v proměnné beaconsDiscoService. Následují deklarace pro charakteristiku Start scan. Charakteristika se v ATT tabulce musí vždy skládat z UUID, oprávnění pro přístup k hodnotě a z hodnoty, kterou charakteristika vystavuje pro čtení případně pro zápis. Dále se charakteristika může skládat z popisu charakteristiky, případně z dalších vlastností. UUID pro charakteristiku Start scan je již deklarované v poli beaconsDiscoScanUUID. Následuje deklarace oprávnění pro přístup k hodnotě charakteristiky. Oprávnění je uložené v proměnné beaconsDiscoScanChar, která obsahuje bitové vyjádření jednotlivých oprávnění. Do této proměnné je tedy uložen bitový součet vyjadřující oprávnění pro čtení i zápis hodnoty charakteristiky Start scan. Dále následuje deklarace proměnné beaconsScanDuration, která obsahuje hodnotu charakteristiky Start scan. Obsah této proměnné v paměti zabírá dva bajty bezznaménkového celočíselného datového typu. Je tedy možné do proměnné beaconsScanDuration uložit hodnotu v rozsahu 0–65535, přičemž tato hodnota vyjadřuje dobu skenování v milisekundách. Každá charakteristika v této práci je opatřená i popisem, aby byla zřejmá odpovědnost charakteristiky. Popis pro charakteristiku Start scan je

deklarovaný v proměnné beaconsDiscoScanCharDesc. V ukázce kódu č. 15 jsou uvedeny zmíněné deklarace.

```
// BEACONS DISCOVERY SERVICE UUIDs
static const uint8_t beaconsDiscoServUUID[ATT_UUID_SIZE] =
{
     TI_BASE_UUID_128(BEACONS_DISCO_SERVICE_UUID)
};

static const uint8_t beaconsDiscoScanUUID[ATT_UUID_SIZE] =
{
     TI_BASE_UUID_128(BEACONS_DISCO_SCAN_UUID)
};
static const gattAttrType_t beaconsDiscoService = {ATT_UUID_SIZE,
                                                   beaconsDiscoServUUID};
static uint8 beaconsDiscoScanChar = GATT_PROP_READ | GATT_PROP_WRITE;
static uint16 beaconsScanDuration = 0;
static  uint8  beaconsDiscoScanCharDesc[]  =  "Start  scan  -  value  =  scan
duration in ms";
```

Ukázka kódu č. 15

Jakmile jsou deklarovány veškeré potřebné části charakteristiky Start scan i služby Beacons discovery, je možné sestavit tabulku vrstvy ATT. Tato tabulka je deklarována jako pole beaconsDiscoServiceAttrTbl struktury gattAttribute_t skládající se ze čtyř hodnot. První hodnota je tvořena strukturou gattAttrType_t, která byla zmíněna výše. Druhá hodnota určuje oprávnění daného atributu. Třetí hodnota je tzv. attribute handle. Tato hodnota je přiřazována interně vrstvou GATT a není tedy specifikována programátorem, tudíž tato hodnota bude nastavená vždy na 0 u každého atributu. Poslední čtvrtá hodnota této struktury je bajtový ukazatel na hodnotu daného atributu. Hodnota, na kterou ukazuje tento ukazatel, může být dlouhá maximálně 512 bajtů.

Na první položce ATT tabulky se nachází atribut specifikující službu Beacons discovery. První položkou tohoto atributu je specifikování UUID vyjadřující, že tento atribut specifikuje primární službu. Toto UUID je dáno specifikací Bluetooth a jeho délka činí 2 bajty. Aby se dala daná služba přečíst, tj. aby Central zařízení mohlo vyčíst hodnotu tohoto atributu, kterým je UUID služby Beacons discovery, musí mít tento atribut oprávnění povolující čtení. Poslední hodnotou tohoto atributu je reference na definované UUID pomocí struktury gattAttrType_t služby

Beacons discovery. Jelikož struktura gattAttribute_t může obsahovat jen referenci na bajt, je reference na strukturu gattAttrType_t přetypována na referenci bajtu.

Druhá položka tabulky ATT obsahuje atribut deklarující charakteristiku. První položkou tohoto atributu je UUID specifikované Bluetooth určující, že následuje deklarace charakteristiky. Tento atribut opět musí mít oprávnění ke čtení. Hodnota tohoto atributu je reference na obsah proměnné beaconsDiscoScanChar obsahující oprávnění pro přístup k hodnotě charakteristiky Start scan.

Třetí položkou ATT tabulky je atribut deklarující samotnou charakteristiku Start scan. První položkou tohoto atributu je reference na pole obsahující UUID definované pro charakteristiku Start scan. Délka tohoto UUID je 16 bajtů. Atribut je oprávněn pro čtení i zápis, přičemž toto oprávnění musí být stejné jako hodnota předchozího atributu. Hodnotou tohoto atributu by měla být reference na samotnou hodnotu charakteristiky Start scan. Ovšem není to nutné a pro účely této práce ani příliš vhodné, jelikož několik hodnot charakteristik závisí na dalších hodnotách a není tedy možné přímo na tomto místě předávat onu referenci na konkrétní hodnotu. Pokud je zde předána reference na hodnotu, je poté tato reference předána při volání R/W callback funkcí profilu.

Poslední čtvrtou položkou tabulky ATT je atribut obsahující popis charakteristiky Start scan. První hodnotou tohoto atributu je UUID definované specifikací Bluetooth, které určuje, že tento atribut obsahuje popis charakteristiky Start scan. Oprávnění tohoto parametru musí být opět pro čtení, aby mohlo Central zařízení přečíst popis charakteristiky. Posledním atributem je předána reference na proměnnou beaconsDiscoScanCharDesc obsahující textový řetězec s popisem charakteristiky.

Jelikož služba Beacons discovery obsahuje pouze jednu charakteristiku Start scan, takto definovaná tabulka vrstvy ATT v poli beaconsDiscoServiceAttrTbl pro službu Beacons discovery je kompletní. V případě, že služba obsahuje více charakteristik, opakují se tyto tři atributy deklarující jednu charakteristiku stále ve stejném pořadí. Deklarace tabulky vrstvy ATT pro službu Beacons list je velice obdobná a nebude proto v textu popsána. Deklarace celé tabulky pro službu Beacons discovery je uvedena v ukázce kódu č. 16.

```
static gattAttribute_t beaconsDiscoServiceAttrTbl[] =
{
     {
          {ATT_BT_UUID_SIZE, primaryServiceUUID},
          GATT_PERMIT_READ,
          0,
          (uint8 *) &beaconsDiscoService
     },
     {
          {ATT_BT_UUID_SIZE, characterUUID},
          GATT_PERMIT_READ,
          0,
          &beaconsDiscoScanChar
     },
     {
          {ATT_UUID_SIZE, beaconsDiscoScanUUID},
          GATT_PERMIT_READ | GATT_PERMIT_WRITE,
          0,
          NULL
     },
     {
          {ATT_BT_UUID_SIZE, charUserDescUUID},
          GATT_PERMIT_READ,
          0,
          beaconsDiscoScanCharDesc
     }
};
```

Ukázka kódu č. 16

6.3.2 Add service funkce

Funkce je zodpovědná za zaregistrování služeb a charakteristik u vrstvy GATT. Tato funkce je volána během inicializace aplikace ve funkci SimpleBLEPeripheral_init. V prvním kroku této funkce je získána reference na strukturu Types_FreqHz zavoláním funkce Timestamp_getFreq. Tato reference je uložená v proměnné freq. Pomocí této reference je později získána frekvence mikroprocesoru, která je použita pro výpočet času pro charakteristiky Age of record a Age of scan. V následujícím kroku je zavolána funkce GATTServApp_RegisterService odpovědná za registraci služeb a charakteristik u vrstvy GATT. V tomto kroku dojde k registraci služby Beacons discovery. Této funkci jsou předány čtyři parametry: ATT tabulka služby Beacons discovery uložená v proměnné beaconsDiscoServiceAttrTbl, počet atributů obsažených v této tabulce získaný makrem GATT_NUM_ATTRS, maximální velikost GATT šifrovacího

klíče a reference na strukturu gattServiceCBs_t obsahující reference na R/W callback funkce profilu. Funkce GATTServApp_RegisterService vrací statusovou hodnotu popisující úspěch či neúspěch při volání této funkce. Tato statusová hodnota je uložena do proměnné scanService. Následuje další volání funkce GATTServApp_RegisterService, které se liší pouze v prvním a druhém parametru, přičemž zde figuruje ATT tabulka služby Beacons list. V posledním kroku se ještě vyhodnotí obsah proměnných scanService a listService. V případě, že obě proměnné obsahují hodnotu SUCCESS, vrátí funkce BeaconsScannerProfile_AddService též hodnotu SUCCESS. V jakémkoli jiném případě vrátí hodnotu FAILURE. Tento rozhodovací blok sloužil především pro ladící účely. Při volání funkce BeaconsScannerProfile_AddService není brán zřetel na tuto návratovou hodnotu ve funkci SimpleBLEPeripheral_init. V ukázce kódu č. 17 je uvedena kompletní deklarace Add service funkce.

```
bStatus_t BeaconsScannerProfile_AddService(void)
{
    Timestamp_getFreq(&freq);

    bStatus_t scanService = GATTServApp_RegisterService(
                            beaconsDiscoServiceAttrTbl,
                            GATT_NUM_ATTRS(beaconsDiscoServiceAttrTbl),
                            GATT_MAX_ENCRYPT_KEY_SIZE,
                            &beaconsScannerProfileCBs);

    bStatus_t listService = GATTServApp_RegisterService(
                            beaconsListServiceAttTbl,
                            GATT_NUM_ATTRS(beaconsListServiceAttTbl),
                            GATT_MAX_ENCRYPT_KEY_SIZE,
                            &beaconsScannerProfileCBs);

    if(scanService == SUCCESS && listService == SUCCESS)
        return SUCCESS;
    else
        return FAILURE;
}
```

Ukázka kódu č. 17

6.3.3 Funkce registrující aplikační callback funkci

Funkce BeaconsScannerProfile_RegisterAppCBs slouží pro zaregistrování aplikační callback funkce u Beacons Scanner Profilu. Tato funkce je volána ihned po zavolání funkce BeaconsScannerProfile_AddService během inicializace aplikace.

V parametru funkce BeaconsScannerProfile_RegisterAppCBs je předána reference na proměnnou typovanou na strukturu beaconsScannerProfileCBs_t z aplikační části. V této proměnné je uložená reference na callback funkci aplikace pro Beacons Scanner Profile. V uvedené funkci se otestuje, zda referenční proměnná appCallbacks obsahuje referenci. Pokud ano, je tato reference uložená do referenční proměnné beaconsScannerProfile_AppCBs nacházející se uvnitř Beacons Scanner Profilu a vrátí se hodnota SUCCESS. Aplikační callback funkce uložená v proměnné beaconsScannerProfile_AppCBs je využívána Beacons Scanner Profilem pro notifikování aplikace, pokud dojde k zápisu hodnoty do charakteristiky Start scan. Podrobnější popis následuje v kapitole **6.3.9 Write callback funkce**. V ukázce kódu č. 18 je možné vidět deklaraci registrační funkce.

```
bStatus_t BeaconsScannerProfile_RegisterAppCBs(beaconsScannerProfileCBs_t
*appCallbacks)
{
    if(appCallbacks)
    {
        beaconsScannerProfile_AppCBs = appCallbacks;

        return SUCCESS;
    }
    else
    {
        return bleAlreadyInRequestedMode;
    }
}
```

Ukázka kódu č. 18

6.3.4 SET funkce

Aby bylo možné přistupovat k hodnotám charakteristik z aplikace, Beacons Scanner Profile obsahuje dvě funkce. Jednou z těchto funkcí je funkce BeaconsScannerProfile_SetParameter sloužící pro nastavení hodnoty charakteristiky. Tato funkce přebírá následující tři parametry: identifikační číslo charakteristiky, délka předávané nové hodnoty a obecný ukazatel na novou hodnotu. Nejprve je nastavena proměnná status na hodnotu SUCCESS, kdy obsah této proměnné je na konci SET funkce vrácen. Pokud proběhne nastavení nové hodnoty namísto původní, obsah proměnné status se nezmění a zůstane hodnota

SUCCESS. Dále následuje rozhodovací blok pomocí konstrukce switch, ve kterém se vyhodnocuje podle přijatého identifikačního čísla charakteristiky. Pokud je při volání funkce BeaconsScannerProfile_SetParameter předáno identifikační číslo charakteristiky, které neexistuje, následuje změna hodnoty proměnné status na hodnotu INVALIDPARAMETER v bloku default. V každém bloku case je pro každou charakteristiku nejprve provedena kontrola velikosti nové hodnoty. Velikost nové hodnoty musí odpovídat velikosti proměnné, ve které je uložena hodnota pro danou charakteristiku. Pokud velikost odpovídá, je nejprve obecný ukazatel přetypován na ukazatel konkrétního datového typu. Poté je provedena dereference tohoto ukazatele, čímž je získána nová hodnota, která je následně uložena namísto původní hodnoty charakteristiky. V jiném případě je obsah proměnné status změněn na hodnotu bleInvalidRange. Pokud jsou předávána pole struktur, dojde k nakopírování dat pomocí funkce memcpy uvnitř konkrétního bloku case. Velikost předávaného pole struktury je kontrolována jako součin velikosti struktury a délky deklarovaného pole struktury. Celá funkce SET je znázorněna v ukázce kódu č. 19.

```
bStatus_t BeaconsScannerProfile_SetParameter(uint8 param, uint16 len, void
*value)
{
    bStatus_t status = SUCCESS;
    switch(param)
    {
        case BEACONS_DISCO_SCAN:
            if(len == sizeof(uint16))
                beaconsScanDuration = *((uint16 *) value);
            else status = bleInvalidRange;
            break;
        case BEACONS_LIST_GET_RECORD:
            if(len == sizeof(uint16))
                beaconsSelectedIndex = *((uint16 *) value);
            else status = bleInvalidRange;
            break;
        case BEACONS_LIST_TOTAL_COUNT:
            if(len == sizeof(uint16))
                beaconsTotalCount = *((uint16 *) value);
            else status = bleInvalidRange;
            break;
        case BEACONS_LIST_ALL_RECORDS:
            if(len == (sizeof(beaconRecord) * BEACONS_RECORDS_LENGTH))
                memcpy(beacons, value,
                           sizeof(beaconRecord) * BEACONS_RECORDS_LENGTH);
            else status = bleInvalidRange;
            break;
        case BEACONS_LIST_MAC_ADDR:
            if(len == (sizeof(macAddr) * BEACONS_MAC_ADDR_LENGTH))
                memcpy(beaconsMacAddr, value,
                           sizeof(macAddr) * BEACONS_MAC_ADDR_LENGTH);
            else status = bleInvalidRange;
            break;
        case BEACONS_LIST_FLAG_OF_MAC:
            if(len == sizeof(uint8))
                beaconsListFlagOfMacValue = *((uint8 *) value);
            else status = bleInvalidRange;
            break;
        case BEACONS_LIST_AGE_OF_SCAN:
            if(len == sizeof(uint32_t))
                beaconsListAgeOfScanValue = *((uint32_t *) value);
            else status = bleInvalidRange;
            break;
        default:
            status = INVALIDPARAMETER;
            break;
    }
    return status;
}
```

Ukázka kódu č. 19

6.3.5 GET funkce

Druhou funkcí pro přístup k hodnotám charakteristik Beacons Scanner Profilu je funkce BeaconsScannerProfile_GetParameter. Tato funkce má jeden parametr, pomocí kterého se předává identifikační číslo charakteristiky. Podle předaného identifikačního čísla charakteristiky funkce vrátí referenci na hodnotu charakteristiky. Pokud je předán neplatný parametr, funkce vrátí hodnotu NULL. Deklarace funkce GET je uvedena v ukázce kódu č. 20.

```
void* BeaconsScannerProfile_GetParameter(uint8 param)
{
    switch(param)
    {
        case BEACONS_DISCO_SCAN:
            return &beaconsScanDuration;
        case BEACONS_LIST_GET_RECORD:
            return &beaconsSelectedIndex;
        case BEACONS_LIST_TOTAL_COUNT:
            return &beaconsTotalCount;
        case BEACONS_LIST_ALL_RECORDS:
            return beacons;
        case BEACONS_LIST_MAC_ADDR:
            return beaconsMacAddr;
        case BEACONS_LIST_FLAG_OF_MAC:
            return &beaconsListFlagOfMacValue;
        case BEACONS_LIST_AGE_OF_SCAN:
            return &beaconsListAgeOfScanValue;
        default:
            return NULL;
    }
}
```

Ukázka kódu č. 20

6.3.6 Způsob ukládání rádiových otisků do paměti MCU

Z důvodu potřeby ukládat co největší počet rádiových otisků, byl zvolen nejúspornější způsob ukládání těchto záznamů. V této podkapitole je popsáno, jak jsou jednotlivé rádiové otisky ukládány do paměti MCU.

Rádiové otisky jsou tvořeny z MAC adresy, RSSI a timestampu. MAC adresa je tvořena 48 bity, tudíž v paměti MCU zabere 6 bajtů. Tento údaj zabere v paměti MCU nejvíce místa. Z tohoto důvodu jsou MAC adresy ukládány do samostatného pole struktury macAddr, přičemž tato struktura obsahuje pole bajtů, jehož délka je právě 6. Toto pole je deklarováno jako beaconsMacAddr a jeho délka činí 256. MAC

adresy jsou v tomto poli ukládány unikátně, je tedy možné uložit rádiové otisky od 256 unikátních zařízení. V paměti MCU toto pole MAC adres zabere 1536 bajtů. Deklarace struktury macAddr je uvedena v ukázce kódu č. 21.

```
typedef struct
{
    uint8 macAddr[B_ADDR_LEN];

} macAddr;
```

Ukázka kódu č. 21

Zbylé údaje rádiového otisku jsou uloženy ve struktuře beaconRecord. Struktura beaconRecord se skládá z následujících tří položek: index MAC adresy uložené v poli beaconsMacAddr, RSSI a času, ve kterém byl rádiový otisk získaný během skenování. Index MAC adresy je uložen v proměnné indexOfMacAddr, přičemž tato proměnná zabere v paměti jeden bajt. Tímto je dán maximální možný počet uložených MAC adres v poli beaconsMacAddr. RSSI je zde uloženo ve stejnojmenné proměnné, jejíž velikost je také jeden bajt. Jedná se o bezznaménkový bajt, i když RSSI hodnota je udávána jako záporné číslo. Je to z důvodu, že hodnota charakteristiky může být pouze bezznaménková. Před uložením RSSI do této struktury je hodnota RSSI vynásobena mínus jedničkou. Po přijetí této bezznaménkové hodnoty v mobilním telefonu je tato hodnota opět vynásobena mínus jedničkou. Již zmíněný čas získání rádiového otisku je ve struktuře uložený ve dvou bajtové proměnné discoTime. Čas v této proměnné je uložen v milisekundách. Jelikož je tato proměnná dvou bajtová, je možné do této proměnné uložit maximální číslo 65535 vyjadřující, že byl záznam objeven v čase skenu 65535 ms. Před uložením této hodnoty do struktury beaconRecord musí být čas přepočítán. Jeden záznam struktury beaconRecord v paměti MCU zabere 4 bajty. S využitím 8 kB cache jako GPRAM, kdy sekce .bss o velikosti 4762 bajtů byla přesunuta z 20 kB SRAM do této GPRAM, je možné do paměti Sensor Tagu uložit 1700 rádiových otisků. Rádiové otisky jsou tedy uloženy v poli beacons struktury beaconRecord, jehož délka je 1700 a v paměti MCU zabere 6800 bajtů. Celková velikost rádiových otisků je součet velikostí polí beaconsMacAddr a beacons, tedy

1536 + 6800 = 8336 bajtů. Deklarace struktury beaconRecord je uvedena v ukázce kódu č. 22.

```
typedef struct
{
    uint8 indexOfMacAddr;
    uint8 rssi;
    uint16 discoTime;

} beaconRecord;
```

Ukázka kódu č. 22

Deklarace výše zmíněných polí uchovávajících rádiové otisky je ukázána v ukázce kódu č. 23.

```
static beaconRecord beacons[BEACONS_RECORDS_LENGTH];
static macAddr beaconsMacAddr[BEACONS_MAC_ADDR_LENGTH];
```

Ukázka kódu č. 23

6.3.7 Ukládání rádiových otisků

Za ukládání rádiových otisků do výše zmíněné struktury je odpovědná funkce BeaconsScannerProfile_AddBeaconRecord. Tato funkce přebírá tři parametry tvořící rádiový otisk, a to MAC adresu, RSSI a timestamp. Nejprve je zavolána funkce BeaconsScannerProfile_FindMacAddr, které je předána MAC adresa. Tato funkce se pokusí zjistit, jestli už je tato MAC adresa obsažena v poli beaconsMacAddr. Pokud se daná MAC adresa nachází v poli beaconsMacAddr, funkce BeaconsScannerProfile_FindMacAddr vrátí index, na kterém se tato MAC adresa nachází. V opačném případě je vrácena hodnota MAC_ADDR_NOT_FOUND. Deklarace funkce BeaconsScannerProfile_FindMacAddr je uvedena v ukázce kódu č. 24.

```
static int16 BeaconsScannerProfile_FindMacAddr(uint8 macAddr[B_ADDR_LEN])
{
    for(uint8 index = 0; index < beaconsMacAddrCount; index++)
    {
        uint8 ret = memcmp(beaconsMacAddr[index].macAddr, macAddr,
                           B_ADDR_LEN);
        if(ret == 0)
            return index;
    }

    return MAC_ADDR_NOT_FOUND;
}
```

Ukázka kódu č. 24

Navrácená hodnota funkce BeaconsScannerProfile_FindMacAddr je poté uložena do proměnné indexOfMacAddr. Následují dvě podmínky testující, zda je možné rádiový otisk uložit. Test první podmínky se skládá ze dvou částí. V první části se testuje, jestli je pole beaconsMacAddr již plné, respektive jestli aktuální počet záznamů v poli beaconsMacAddr je roven délce tohoto pole. Tento test ovšem není dostačující, neboť pokud je nalezen index MAC adresy v tomto poli, není potřeba přidávat nový záznam MAC adresy a je možné sestavit rádiový otisk. Z tohoto důvodu obsahuje test této podmínky ještě druhou nezbytnou část. V této části se testuje, zda obsah proměnné indexOfMacAddr je roven hodnotě MAC_ADDR_NOT_FOUND. Pokud tedy při přidávání nového rádiového otisku je pole MAC adres plné a zároveň MAC adresa není obsažena v poli MAC adres, není možné sestavit konzistentní rádiový otisk a je vykonán blok kódu uvnitř této podmínky. Dojde k nastavení příznaku charakteristiky Flag na hodnotu 1 a rádiový otisk je zahozen. Ve druhé podmínce se kontroluje, zda není pole beacons plné rádiových otisků. Pokud je počet rádiových otisků roven délce tohoto pole, je rádiový otisk zahozen. Pokud není splněna žádná z těchto předchozích dvou podmínek, dojde k uložení rádiového otisku do paměti MCU.

V první řadě se vykonají nezbytné výpočty s časem. Tedy je potřeba uložit do položky discoTime struktury beaconRecord čas pořízení rádiového otisku během skenování, tj. ve kterém čase skenu byl záznam pořízen. Nejprve je nutné vypočítat deltu charakterizovanou jako rozdíl aktuálního časového razítka pořízení rádiového otisku a časového razítka startu skenu, které je uložené v proměnné

beaconsListAgeOfScanValue jako hodnota charakteristiky Age of scan. Tento rozdíl je ještě vynásoben hodnotou 1000, aby byl výsledný čas v milisekundách. Poté je hodnota proměnné delta vydělena frekvencí, s jakou běží samotné MCU. Tato frekvence je získána skrze referenci na strukturu Types_FreqHz, přičemž frekvence se nachází v proměnné lo uvnitř této struktury. Výsledný čas v milisekundách je uložen ve 4 bajtové proměnné time.

Nyní následuje sestavení rádiového otisku. Rádiové otisky jsou uloženy v poli beacons. Rádiový otisk se ukládá vždy na následující index, jehož hodnota je rovna proměnné beaconsTotalCount. Tento čítač je vždy po přidání rádiového otisku zvýšen. Nejprve je uložen vypočítaný čas nacházející se v proměnné time. Proměnná time je 4 bajtová, zatímco proměnná discoTime struktury beaconRecord pouze 2 bajtová. Na tomto řádku, kdy se obsah proměnné time zkopíruje do proměnné discoTime, dojde k oříznutí vyšších 2 bajtů proměnné time. Tato operace je v pořádku, neboť dojde k uložení času v rozsahu 0 ms až 65535 ms. Následuje uložení hodnoty RSSI, která je před uložením vynásobena mínus jedničkou z důvodu zmíněného výše v textu. Poté následuje poslední krok, kterým je spárování indexu MAC adresy z pole beaconsMacAddr s rádiovým otiskem uloženým v poli beacons. Tento úkon je proveden na základě hodnoty proměnné indexOfMacAddr. Pokud je již MAC adresa uložená v poli beaconsMacAddr, proměnné indexOfMacAddr obsahuje index této MAC adresy, tj. obsah proměnné indexOfMacAddr je různý od hodnoty MAC_ADDR_NOT_FOUND. V tomto případě se vykoná blok kódu uvnitř větve if, tedy index MAC adresy je uložen do stejnojmenné proměnné uvnitř struktury beaconRecord a hodnota proměnné beaconsTotalCount je zvýšena o jedničku. V opačném případě je vykonán blok kódu uvnitř větve else. Nejprve se překopíruje MAC adresa do pole beaconsMacAddr na aktuální index pomocí funkce memcpy. I zde platí, že aktuální počet záznamů pole se rovná následujícímu indexu nové hodnoty. Jakmile je MAC adresa zkopírována, nastaví se index MAC adresy rádiovému otisku. Po této operaci musí být zvýšeny obsahy proměnných beaconsTotalCount a beaconsMacAddrCount o jedničku. V ukázce kódu č. 25 je uvedena deklarace funkce odpovědné za přidávání rádiových otisků.

```
void BeaconsScannerProfile_AddBeaconRecord(uint8 macAddr[B_ADDR_LEN], int8
rssi, uint32_t timestamp)
{
    int16 indexOfMacAddr = BeaconsScannerProfile_FindMacAddr(macAddr);

    if(beaconsMacAddrCount == BEACONS_MAC_ADDR_LENGTH &&
       indexOfMacAddr == MAC_ADDR_NOT_FOUND)
    {
        beaconsListFlagOfMacValue = 1;
        return;
    }

    if(beaconsTotalCount == BEACONS_RECORDS_LENGTH)
        return;

    uint32_t delta = (timestamp - beaconsListAgeOfScanValue) * 1000; //ms
    uint32_t time = delta / freq.lo;

    beacons[beaconsTotalCount].discoTime = time;
    beacons[beaconsTotalCount].rssi = -1 * rssi;

    if(indexOfMacAddr != MAC_ADDR_NOT_FOUND)
    {
        beacons[beaconsTotalCount++].indexOfMacAddr = indexOfMacAddr;
    }
    else
    {
        memcpy(beaconsMacAddr[beaconsMacAddrCount].macAddr, macAddr,
               B_ADDR_LEN);
        beacons[beaconsTotalCount++].indexOfMacAddr = beaconsMacAddrCount++;
    }
}
```

Ukázka kódu č. 25

6.3.8 Read callback funkce

Tato callback funkce je volána BLE stackem, jakmile Central zařízení pošle požadavek na čtení hodnoty určité charakteristiky v případě, že má daná charakteristika oprávnění pro čtení. Při volání této callback funkce BLE stack předá 7 parametrů. V textu jsou popsány pouze využívané parametry. Prvním parametrem je reference na strukturu gattAttribute_t v proměnné pAttr. V této referenční proměnné je předána BLE stackem konkrétní charakteristika, na kterou přišel požadavek na data. Druhým parametrem je adresa paměti uložená v proměnné pValue, na kterou se zkopíruje obsah charakteristiky. Hodnotu z této adresy v paměti poté BLE stack odešle Central zařízení, které o tato data žádalo.

Třetím parametrem je adresa v paměti uložená v proměnné pLen, na kterou se uloží délka dat v bajtech, které jsou poslány Central zařízení.

Nejprve je deklarována proměnná status, do které se uloží hodnota SUCCESS. Obsah této proměnné je změněn, jakmile nastane situace, během které nedojde k vrácení hodnot z požadované charakteristiky. Obsah této proměnné je na konci read callback funkce vrácen. V první řadě se kontroluje, zda délka UUID charakteristiky odpovídá 16 bajtům. Pokud by nebyla délka UUID charakteristiky dlouhá 16 bajtů, je vykonán blok kódu ve větvi else. V této větvi se nastaví délka hodnoty na 0 bajtů a do proměnné status je uložena hodnota ATT_ERR_ATTR_NOT_FOUND, protože v tomto profilu není využívána žádná standardní charakteristika, která by měla UUID dlouhé 2 bajty. Po této kontrole je pomocí makra BUILD_UINT16 sestaven zkrácený tvar UUID charakteristiky a uložen do proměnné uuid.

Následuje konstrukce switch, ve které se rozhoduje na základě hodnoty proměnné uuid. Pokud by zde nebyla nalezena shoda s UUID definovaných charakteristik, bude vykonán blok default, ve kterém proběhne stejná činnost, jako ve větvi else. Tedy délka hodnoty je nastavena na hodnotu 0 bajtů a do proměnné status je uložena hodnota ATT_ERR_ATTR_NOT_FOUND. Každá charakteristika zde má svůj blok case, ve kterém vrátí svou hodnotu. V každém bloku case je skrze referenční proměnnou pLen nastavena délka hodnoty charakteristiky v bajtech.

První blok case BEACONS_DISCO_SCAN_UUID náleží charakteristice Star scan. Pomocí funkce memcpy je zkopírován obsah proměnné beaconsScanDuration, ve které je uložena hodnota charakteristiky Start scan.

Druhý blok case BEACONS_LIST_FLAG_OF_MAC_UUID náleží charakteristice Flag. Při deklaraci atributu této charakteristiky byla předána reference na hodnotu charakteristiky Flag. V tomto druhém bloku case je využito toho, že reference na hodnotu charakteristiky Flag je získána z reference na atribut charakteristiky Flag. Vzhledem k tomu, že hodnota charakteristiky Flag je dlouhá 1 bajt, není nutné využívat funkci memcpy.

Ve třetím bloku case BEACONS_LIST_TOTAL_COUNT_UUID je vrácena hodnota charakteristiky Total count. Obsah proměnné beaconsTotalCount je zkopírován pomocí funkce memcpy. Následují tři bloky case pro tři charakteristiky

tvořící jeden rádiový otisk. Aby bylo možné rozumně vyčíst všechny rádiové otisky, je zde charakteristika Set index. Skrze tuto charakteristiku se nastaví index rádiového otisku, který lze následně vyčíst skrze následující tři charakteristiky. Index charakteristiky Set index je uložen v proměnné beaconsSelectedIndex.

První charakteristikou rádiového otisku je charakteristika MAC address v bloku case BEACONS_LIST_MAC_ADDR_UUID. Nejprve je potřeba získat index do pole MAC adres aktuálního rádiového otisku. Pro lepší čitelnost kódu je tento index uložen do proměnné index. Poté je pomocí funkce memcpy zkopírována MAC adresa nacházející se na indexu uloženém v proměnné index z pole MAC adres.

Druhou charakteristikou rádiového otisku je charakteristika RSSI v bloku case BEACONS_LIST_RSSI_UUID. Bezznaménková hodnota RSSI aktuálního rádiového otisku je zde zkopírována na adresu paměti předanou BLE stackem.

Třetí charakteristika rádiového otisku je charakteristika Age of record v bloku case BEACONS_LIST_AGE_UUID. Zde je opět pomocí funkce memcpy zkopírován čas pořízení aktuálního rádiového otisku.

Poslední charakteristikou určenou pro čtení je charakteristika Age of scan v bloku case BEACONS_LIST_AGE_OF_SCAN_UUID. V tomto bloku kódu proběhne téměř identický výpočet s časem, jako při ukládání časového údaje rádiového otisku. Nejprve je získáno aktuální časové razítko pomocí funkce Timestamp_get32, které je poté uloženo do proměnné actualTime. Dále proběhne výpočet delty, tj. rozdíl mezi aktuálním časovým razítkem a časovým razítkem uloženým při začátku skenování do proměnné beaconsListAgeOfScanValue. Protože i tato charakteristika vrací čas v milisekundách, je rozdíl časových razítek před uložením do proměnné delta vynásoben hodnotou 1000. Následně je delta vydělena frekvencí procesoru a tím je získána doba od zahájení skenování. Hodnota získaná dělením je uložena do 4 bajtové proměnné time. Poté jsou funkcí memcpy zkopírovány první 2 bajty proměnné time, tj. nejnižší bajty. Tím je docíleno, že doba od zahájení skenování je v rozmezí 0 ms až 65535 ms, stejně jako je tomu u charakteristiky Age of record. Deklarace read callback funkce je uvedena v ukázce kódu č. 26.

```
static bStatus_t beaconsScannerProfileReadAttrCB(uint16_t connHandle,
gattAttribute_t *pAttr, uint8_t *pValue, uint16_t *pLen, uint16_t offset,
uint16_t maxLen, uint8_t method)
{
    bStatus_t status = SUCCESS;
    if(pAttr->type.len == ATT_UUID_SIZE) {
        uint16 uuid = BUILD_UINT16(pAttr->type.uuid[12],
                                   pAttr->type.uuid[13]);
        switch(uuid) {
            case BEACONS_DISCO_SCAN_UUID:
                *pLen = BEACONS_SCAN_LENGTH;
                memcpy(pValue, &beaconsScanDuration, BEACONS_SCAN_LENGTH);
                break;
            case BEACONS_LIST_FLAG_OF_MAC_UUID:
                *pLen = 1;
                pValue[0] = *pAttr->pValue;
                break;
            case BEACONS_LIST_TOTAL_COUNT_UUID:
                *pLen = BEACONS_TOTAL_COUNT_LENGTH;
                memcpy(pValue, &beaconsTotalCount,
                       BEACONS_TOTAL_COUNT_LENGTH);
                break;
            case BEACONS_LIST_MAC_ADDR_UUID:
                *pLen = B_ADDR_LEN;
                uint8 index = beacons[beaconsSelectedIndex].indexOfMacAddr;
                memcpy(pValue, beaconsMacAddr[index].macAddr, B_ADDR_LEN);
                break;
            case BEACONS_LIST_RSSI_UUID:
                *pLen = 1;
                pValue[0] = beacons[beaconsSelectedIndex].rssi;
                break;
            case BEACONS_LIST_AGE_UUID:
                *pLen = BEACONS_AGE_OF_RECORD_LENGTH;
                memcpy(pValue, &beacons[beaconsSelectedIndex].discoTime,
                       BEACONS_AGE_OF_RECORD_LENGTH);
                break;
            case BEACONS_LIST_AGE_OF_SCAN_UUID:
                *pLen = BEACONS_AGE_OF_SCAN_LENGTH;
                uint32_t actualTime = Timestamp_get32();
                uint32_t delta=(actualTime-beaconsListAgeOfScanValue)*1000;
                uint32_t time = delta / freq.lo;
                memcpy(pValue, &time, BEACONS_AGE_OF_SCAN_LENGTH);
                break;
            default:
                *pLen = 0;
                status = ATT_ERR_ATTR_NOT_FOUND;
                break;
        }
    } else {
        *pLen = 0;
        status = ATT_ERR_ATTR_NOT_FOUND;
    } return status;
}
```

Ukázka kódu č. 26

6.3.9 Write callback funkce

Callback funkce beaconsScannerProfileWriteAttrCB slouží pro zápis dat do charakteristik. Tato callback funkce je též volána BLE stackem poté, co BLE stack zkontroluje oprávnění dané charakteristiky. Aby BLE stack zavolal tento callback, musí mít charakteristika oprávnění pro zápis. V tomto callbacku figuruje 7 stejných parametrů, jako tomu je u read callback funkce. Zde je jen drobný rozdíl v referenční proměnné pValue a v proměnné len. Při volání tohoto callbacku BLE stack předá referenci v proměnné pValue na hodnotu přijatou od Central zařízení. V proměnné len je poté obsažena délka této nové přijaté hodnoty. V této callback funkci probíhá zpracování velice podobně, jako je tomu u read callback funkce.

Nejprve je deklarována proměnná status s počáteční hodnotou SUCCESS. I zde se změní obsah proměnné status v případě, kdy nedojde k úspěšnému zapsání nové hodnoty do dané charakteristiky. Na konci write callbacku je hodnota proměnné status vrácena BLE stacku.

Dále je deklarována proměnná notifyApp a inicializována hodnotou 255. Do této proměnné je při zápisu nové hodnoty do charakteristiky uloženo identifikační číslo této charakteristiky. Na konci tohoto callbacku je poté před vrácením hodnoty status podmínka testující, zda by měl být zavolán aplikační callback. Test této podmínky je složen ze tří částí, které musí platit současně. V první části testu se zjišťuje, jestli je obsah proměnné notifyApp různý od hodnoty 255, tj. jestli obsahuje některé identifikační číslo charakteristiky. Druhá část testu kontroluje, zda je v proměnné beaconsScannerProfile_AppCBs reference na strukturu, obsahující callback aplikace. A třetí část podmínky testuje, zda je v této struktuře obsažena reference na callback funkci aplikace. Pokud platí tato tři kritéria testu v podmínce, je zavolána aplikační callback funkce, kdy v parametru funkce je předáno identifikační číslo charakteristiky, u které byla zapsána nová hodnota.

Po deklaraci proměnné notifyApp následuje podmínka, která testuje délku UUID předané charakteristiky BLE stackem. I zde musí být délka UUID 16 bajtů. Pokud není, je vykonán blok kódu ve větvi else, kde se změní obsah proměnné status na hodnotu ATT_ERR_INVALID_HANDLE. V opačném případě se pokračuje

větví if. V prvním kroku je opět nutné sestavit zkrácený tvar UUID pomocí makra BUILD_UINT16. Zkrácené UUID je poté uloženo ve stejnojmenné proměnné.

Následuje konstrukce switch, kde se rozhoduje podle UUID charakteristiky. Charakteristiky s oprávněním pro zápis jsou v této práci pouze dvě. Každá z těchto dvou charakteristik má zde svůj case blok. Pokud by přišla charakteristika s 16 bajtovou délkou UUID, která by se nespárovala se žádným case blokem, vykoná se blok default. V tomto bloku dojde k nastavení obsahu proměnné status na hodnotu ATT_ERR_ATTR_NOT_FOUND. V každém case bloku je provedena ještě kontrola délky nově přijaté hodnoty. Nejprve se kontroluje hodnota offset. Pokud je hodnota offsetu rovna 0, zkontroluje se hodnota proměnné len. V případě, že se hodnota proměnné len nerovná délce hodnoty charakteristiky, je obsah proměnné status změněn na hodnotu ATT_ERR_INVALID_VALUE_SIZE.

První case blok BEACONS_DISCO_SCAN_UUID náleží charakteristice Start scan. Po kontrole přijaté hodnoty v případě, že je obsah proměnné status roven hodnotě SUCCESS následuje zkopírování nové hodnoty pomocí funkce memcpy do proměnné scanDuration. Tímto krokem dojde k sestavení hodnoty z jednotlivých bajtů. Následuje otestování této sestavené hodnoty v proměnné scanDuration. Pokud je nová hodnota proměnné scanDuration větší než 0, tj. proměnná scanDuration obsahuje hodnotu určující dobu skenování a zároveň je obsah proměnné beaconsScanDuration roven 0, tj. neprobíhá skenování, je tato nová hodnota nastavena do proměnné beaconsScanDuration charakteristiky Start scan. Poté je ještě nastaven obsah proměnné notifyApp na hodnotu identifikačního čísla charakteristiky Start scan, tedy na hodnotu BEACONS_DISCO_SCAN.

Druhý case blok BEACONS_LIST_GET_RECORD_UUID patří charakteristice Set index. Zde je průběh velice obdobný, jako u charakteristiky Start scan. Nejprve je sestaven index z přijaté hodnoty pomocí funkce memcpy. Následuje porovnání, kdy hodnota indexu musí být menší než hodnota charakteristiky Total count vyjadřující celkový počet získaných rádiových otisků. Pokud je splněna daná podmínka, je tato nově přijatá hodnota uložena do proměnné beaconsSelectedIndex charakteristiky Set index. V ukázce kódu č. 27 je uvedena deklarace write callback funkce.

```
static bStatus_t beaconsScannerProfileWriteAttrCB(uint16_t connHandle,
gattAttribute_t *pAttr, uint8_t *pValue, uint16_t len, uint16_t offset,
uint8_t method)
{
    bStatus_t status = SUCCESS;
    uint8 notifyApp = 0xFF;
    if(pAttr->type.len == ATT_UUID_SIZE) {
        uint16 uuid = BUILD_UINT16(pAttr->type.uuid[12],
                                   pAttr->type.uuid[13]);
        switch(uuid) {
            case BEACONS_DISCO_SCAN_UUID:
                if(offset == 0) {
                    if(len != BEACONS_SCAN_LENGTH)
                        status = ATT_ERR_INVALID_VALUE_SIZE;
                } else status = ATT_ERR_ATTR_NOT_LONG;

                if(status == SUCCESS) {
                    uint16 scanDuration;
                    memcpy(&scanDuration, pValue, BEACONS_SCAN_LENGTH);

                    if(scanDuration > 0 && beaconsScanDuration == 0) {
                        beaconsScanDuration = scanDuration;
                        notifyApp = BEACONS_DISCO_SCAN;
                    }
                }
                break;
            case BEACONS_LIST_GET_RECORD_UUID:
                if(offset == 0) {
                    if(len != BEACONS_TOTAL_COUNT_LENGTH)
                        status = ATT_ERR_INVALID_VALUE_SIZE;
                } else status = ATT_ERR_ATTR_NOT_LONG;

                if(status == SUCCESS) {
                    uint16 index;
                    memcpy(&index, pValue, BEACONS_TOTAL_COUNT_LENGTH);

                    if(index < beaconsTotalCount)
                        beaconsSelectedIndex = index;
                }
                break;
            default:
                status = ATT_ERR_ATTR_NOT_FOUND;
                break;
        }
    } else status = ATT_ERR_INVALID_HANDLE;

    if(notifyApp != 0xFF && beaconsScannerProfile_AppCBs &&
       beaconsScannerProfile_AppCBs->pfnBeaconsScannerProfileChange)
    beaconsScannerProfile_AppCBs->pfnBeaconsScannerProfileChange(notifyApp);

    return status;
}
```

Ukázka kódu č. 27

7 Výsledky a testování

Navzdory všem překážkám se povedlo zhotovit prototyp přenosného zařízení na platformě System-on-Chip CC2650 Sensor Tag od Texas Instruments. Toto prototypové zařízení dokáže uložit rádiové otisky od 256 různých zařízení, přičemž celkový počet uložených rádiových otisků může být až 1700. Sensor Tag může skenovat po dobu v rozsahu 1 ms až 65535 ms, kdy hodnota v tomto rozsahu je zaslána Sensor Tagu při zápisu do charakteristiky umožňující zahájení skenu.

Pro účely této práce vznikla mobilní aplikace pro Android, sloužící ke komunikaci mezi smartphonem a Sensor Tagem. Tato aplikace umožňuje zahájení skenování Sensor Tagu na dobu specifikovanou uživatelem. Jakmile Sensor Tag dokončí skenování, tato aplikace vyčte veškerá data ze Sensor Tagu a poté tato data vyexportuje do souboru ve formátu JSON. Takto vyexportovaná data jsou uložena v telefonu ve složce SensorTag, odkud jsou přístupná pro další zpracování.

Byla otestována výdrž baterie v aktivním režimu Sensor Tagu. Pro tento test byl napsán testovací program, který opakovaně spouštěl skenování po dobu 25 sekund. Poté vždy vyčetl veškerá data získaná Sensor Tagem během skenování. V průběhu tohoto testu byly přítomny 3 zařízení iBeacon, přičemž průměrný počet zachycených advertise paketů Sensor Tagem byl zhruba 200 až 400. Za těchto podmínek vydržela nová baterie přibližně 12 hodin a bylo provedeno cca 470 skenů.

Během pokusů při skenování Sensor Tagem bylo využito také zařízení Raspberry Pi 3 Model B. To přispělo ke zjištění, že během pasivního skenování zařízení iBeacon vysílají advertise pakety v daleko delších intervalech, než jaké mají nastaveny. Tento jev je zapříčiněn úsporou energie zařízení iBeacon. K tomuto zjištění bylo využito pasivní skenování pomocí příkazu hcitool v následujícím tvaru v ukázce příkazu č. 1.

sudo hcitool lescan --duplicates --passive

Ukázka příkazu č. 1

Tento příkaz je nutné spustit s právy roota. Z tohoto důvodu je zde uveden příkaz sudo. Příkaz lescan určuje, že se bude skenovat v Low Energy módu.

Parametr duplicates udává, že budou do konzole vypsány všechny zachycené advertise pakety. Uvedením parametru passive je docíleno pasivního skenování. Pokud by nebyl tento parametr uveden, implicitně je nastaveno skenování aktivní. Při spuštění tohoto příkazu bylo možné pozorovat, že advertise pakety vysílané iBeacon zařízeními byly vysílány zhruba v intervalu 2 až 3 sekundy. Naopak při aktivním skenování již nebylo možné rozeznat interval vysílaných advertise paketů.

8 Závěr

Stanovené cíle práce se podařilo zdárně naplnit. Na platformě System-on-Chip CC2650 Sensor Tag od Texas Instruments byla úspěšně realizována aplikace prototypového zařízení sloužícího pro experimenty se zpřesňováním indoor lokalizace. Toto prototypové zařízení je schopno uložit rádiové otisky od 256 různých zařízení, kdy celkový počet rádiových otisků může být až 1700. Prototypové zařízení může provádět skenování okolních iBeacon zařízení v časovém intervalu od 1 ms až do 65535 ms. Délka skenování závisí na hodnotě zapsané smartphonem do charakteristiky Sensor Tagu odpovědné za zahájení skenu. Baterie v Sensor Tagu vydrží přibližně 12 hodin při aktivním používání. To odpovídá cca 470 skenům, přičemž každý trvá 25 sekund. Sensor Tag je možné používat s LCD displejem, ale také bez něj. Pokud je využíván LCD displej, je možné vidět aktuální bluetooth stav Sensor Tagu, respektive zda je Sensor Tag v advertise módu nebo je připojen k jinému Central zařízení. Dále je na displeji zobrazeno, zda probíhá skenování, počet aktuálně přijatých advertise paketů od iBeacon zařízení a přijatá hodnota trvání skenování v milisekundách. Pokud by se z nějakého důvodu nepovedlo spustit skenování, je na displej vypsána tato skutečnost včetně hodnoty chybového stavu.

Zdrojové kódy firmwaru pro Sensor Tag jsou volně dostupné z githubu[6]. Projekt lze zkompilovat pomocí volně dostupných nástrojů. Tento projekt je určen pro IDE Code Composer Studio, přičemž byl vytvořen ve verzi 7.2.0.00013.

V neposlední řadě byla napsána aplikace pro Android. Tato aplikace umožňuje spouštět skenování Sensor Tagu na dobu stanovenou uživatelem. Po dokončení skenování aplikace vyčte veškerá data získaná během skenování. Poté jsou tato data aplikací vyexportována do souboru ve formátu JSON. Tyto soubory se nacházejí ve složce SensorTag, odkud jsou přístupné pro další zpracování.

[6] https://github.com/kacer/cc2650-indoor-localization

9 Seznam použité literatury

[1] Bluetooth SIG, *Bluetooth Core Specification* [online]. [cit. 2017-07-02]. Dostupné z: https://www.bluetooth.com/specifications/bluetooth-core-specification

[2] Bluetooth SIG, *Bluetooth Core Specification – Adopted specifications* 2016 [cit. 2017-07-02]. Dostupné z: https://www.bluetooth.org/en-us/specification/adopted-specifications

[3] TOWNSEND, Kevin. *GATT | Introduction to Bluetooth Low Energy* [online]. 2015-05-04 [cit. 2017-07-02]. Dostupné z: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

[4] *Introduction to Bluetooth LE* [online]. [cit. 2017-07-02]. Dostupné z: https://github.com/tigoe/BLEDocs/wiki/Introduction-to-Bluetooth-LE

[5] SimpleLink™ multi-standard CC2650 SensorTag™ kit reference design. *Texas Instrumets* [online]. [cit. 2018-01-28]. Dostupné z: http://www.ti.com/tool/TIDC-CC2650STK-SENSORTAG?keyMatch=cc2650stk&tisearch=Search-EN-Products

[6] Simplelink SensorTag - TI.com. *Texas Instruments* [online]. [cit. 2018-01-28]. Dostupné z: http://www.ti.com/ww/en/wireless_connectivity/sensortag/tearDown.html

[7] CC2650 SimpleLink multi-standard 2.4 GHz ultra-low power wireless MCU | TI.com. *Texas Instruments* [online]. [cit. 2018-01-28]. Dostupné z: http://www.ti.com/product/CC2650/description

[8] CC2650 SimpleLink™ Multistandard Wireless MCU datasheet (Rev. B). *Texas Instruments* [online]. ©2017 [cit. 2018-01-30]. Dostupné z: http://www.ti.com/lit/ds/symlink/cc2650.pdf

[9] BLUEPIXEL TECHNOLOGIES LLP. BLE Scanner. In: *Google Play* [online]. ©2018 [cit. 2018-02-01]. Dostupné z: https://lh3.googleusercontent.com/sYN-3WgK_PRVGQOFhb562VrHsOCWrJXVbyWfd9LAW_tYbRcJZja4PGbY27HMPhmwbRiG=h310

[10] Texas Instruments SensorTag CC2650 (11). In: *IoTool* [online]. [cit. 2018-02-01]. Dostupné z: https://iotool.io/extensions/sensors/texas-instruments-sensortag-cc2650-environmental-sensor-11

[11] Estimote Beacons Sketch Resource. In: *Sketch App Sources* [online]. ©2012-2018 [cit. 2018-02-01]. Dostupné z: https://www.sketchappsources.com/free-source/1297-estimote-beacon-sketch-freebie-resource.html

[12] Signal Icon. In: *Pvhc* [online]. July 28, 2017 [cit. 2018-02-01]. Dostupné z: http://www.pvhc.net/Signal-Icon20vqvgmmgl/

[13] CC2640/CC2650 Bluetooth low energy Software Developer's Guide (Rev. E). *Texas Instruments* [online]. ©2018 [cit. 2018-03-21]. Dostupné z: http://www.ti.com/lit/ug/swru393e/swru393e.pdf

10 Přílohy

Obsah přiloženého CD:

- Adresář Sensor Tag obsahuje 2 podadresáře:
    - V adresáři cc2650-indoor-localization-master se nachází samotný projekt firmwaru pro Sensor Tag se zdrojovými kódy, které byly popsány v bakalářské práci.
    - V adresáři ble_examples-ble_examples-2.2 se nachází všechny příklady stažené z githubu Texas Instruments. V těchto příkladech se nachází projekt simple_peripheral_observer, na kterém je založena aplikace Sensor Tagu.
- Adresář Android obsahuje vyexportovanou aplikaci ve formátu .apk a složku src, ve které se nachází zdrojové kódy mobilní aplikace pro Android.

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Donát Martin	Orebitská 288, Třebechovice pod Orebem	I1500350

TÉMA ČESKY:

Vývoj periferie s využitím Bluetooth Low Energy SoC

TÉMA ANGLICKY:

Development of Bluetooth Low Energy Peripheral based on a SoC

VEDOUCÍ PRÁCE:

Ing. Pavel Kříž, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl: Analyzovat možnosti vývoje Bluetooth Low Energy (BLE) periferie pomocí System-on-Chip řešení Texas Instruments CC2650. Navrhnout a implementovat konkrétní aplikaci periferie komunikující se smartphonem. Úkolem periferie bude naskenovat okolní BLE zařízení a zaslat jejich seznam do smartphonu.

Osnova:
1. Úvod
2. Cíl
3. Technologie Bluetooth Low Energy (BLE)
4. Implementace BLE na platformě Texas Instruments Simplelink
5. Analýza a návrh řešení
6. Implementace
7. Výsledky a testování
8. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

1) Texas Instruments: CC2640 and CC2650 SimpleLink Bluetooth Low Energy Software Stack 2.2.1 Developer's Guide

2) Bluetooth Core Specification https://www.bluetooth.com/specifications/bluetooth-core-specification

Podpis studenta: Donát ..... **Datum:** 10.10.2017

Podpis vedoucího práce: ..... **Datum:** 10.10.17