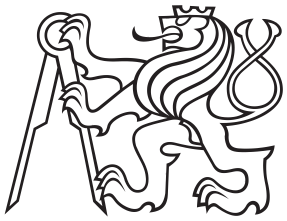
Diplomová práca



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra mikroelektroniky

Bezdrátový update firmwaru na procesoru TI CC1310

Bc. Richard Kučkovský

Vedúci práce: Ing. Vladimír Janíček, Ph.D
Odbor: Elektronika
Máj 2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kučkovský** Jméno: **Richard** Osobní číslo: **475405**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra mikroelektroniky**

Studijní program: **Elektronika a komunikace**

Studijní obor: **Elektronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Bezdrátový upgrade firmwaru na procesoru TI CC1310

Název diplomové práce anglicky:

Wireless Firmware Upgrade on TI CC1310 CPU

Pokyny pro vypracování:

1) Proveďte analýzu trhu z pohledu způsobů provedení upgradu softwaru embedded procesorů.
2) Navrhněte proces upgradu pomocí bezdrátové komunikace (OTA) s důrazem na minimalizaci spotřeby elektrické energie koncového bezdrátového bodu (node).
3) Realizujte software pro obsluhu tohoto procesu.
4) Ověřte funkčnost na modelu hubu s koncovými bezdrátovými zařízeními.
5) Proveďte analýzu přínosů dané metody a porovnejte s konvenčními metodami upgradu FW.

Seznam doporučené literatury:

1) webstránky TI.com
2) Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking, ASIN: B00K1N23LA

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Vladimír Janíček, Ph.D., katedra mikroelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **13.02.2019** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **20.09.2020**

Ing. Vladimír Janíček, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

14. 5. 2019 — Datum převzetí zadání

Podpis studenta

Poďakovanie

Ďakujem môjmu vedúcemu diplomovej práce Ing. Vladimírovi Janíčkovi, Ph.D za užitočné pripomienky, ochotu a usmernenie pri písaní diplomovej práce. Ďalej by som sa chcel poďakovať mojim kolegom Tomášovi Bartákovi a Maksymovi Hukovi za ich cenné rady a možnosť pracovať na tomto projekt. V neposlednom rade patrí veľká vďaka mojej rodine a priateľom bez ktorých podpory by som to nedokázal.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne, a že som uviedol použitú literatúru.

V Prahe, 24. mája 2019

Abstrakt

Vývoj funkcionality vstavaných zariadení rýchlo napreduje a s ohľadom na tento fakt je potrebné efektívne a jednuducho ich aktualizovať. Aby to bolo možné dosiahnuť, táto práca sa zaoberá s implementáciou bezdrôtovej aktualizácie firmvéru mikrokontroléra TI CC1310. Najskôr je zistený aktuálny stav možností aktualizácie na trhu, po ktorom prebieha zoznámenie s použitou hardvérovou a softvérovou platformou. Po príprave je navrhnutý a následne implementovaný koncept riešenia, ktorý je na záver otestovaný a zhodnotený.

Kľúčové slová: CC1310, OTA, firmware, update

Vedúci práce: Ing. Vladimír Janíček, Ph.D

Abstract

The functionality of embedded devices is evolving quickly, and that is why it is needed to upgrade their firmware simply and effectively. To do that, over-the-air (OTA) update is often used. This master's thesis deals with the OTA update for the TI CC1310 microcontroller. Thesis starts with the analysis of current ways of upgrading the firmware and description of used hardware and software. After that the solution draft is proposed, developed, and the final implementation is evaluated.

Keywords: CC1310, OTA, firmware, update

Title translation: Wireless Firmware Upgrade on TI CC1310

Obsah

Obrázky

Tabuľky

Kapitola 1

Úvod

Využitie drobnej elektroniky sa za posledné roky vďaka väčšiemu výpočtovému výkonu pri nízkej spotrebe rozvíja rapídnym tempom. Automatizácia jej funkcie v spojení s pripojením do siete Internet sa pretavilo do definície pojmu Internet of Things (IoT), ktorý má do budúcnosti ohromný potenciál.

Koncept inteligentných budov je jedným z mnohých príkladov platformy Internetu vecí. Vrámci komplexného riešenia sa môžu využívať na zber dát rôzneho charakteru jednotné moduly, ktoré sa líšia svojou softvérovou výbavou. Moduly tak môžu zbierať dáta za pomoci rôznych senzorov, ako je teplota a vlhkosť, alebo ako senzory pre obsadenie parkovacích miest, kancelárií a zasadacích miestností.

Na firmvére modulov je potrebné neustále pracovať - implementujú sa rôzne rozšírenia funkčnosti, alebo sa opravujú chyby z predošlej verzie. Nevýhodou doterajšieho systému je ale nepraktický spôsob aktualizácie týchto modulov. Tá sa môže vykonať iba rozhraním USB s počítačom obsahujúcim príslušný softvér na aktualizáciu a nový obraz firmvéru.

Ucelené riešenie pre jednu budovu ale môže obsahovať veľké množstvo modulov, ktoré sa nachádzajú na často neprístupných miestach, niektoré sú dokonca aj napevno pripevnené a nie je možné s nimi hýbať. Okrem ťažkej prístupnosti k modulom sa zákazníci využívajúci riešený koncept môžu nachádzať v rôznych častiach sveta, čo robí riadenú aktualizáciu firmvéru kvalifikovaným technikom ešte náročnejšie a zároveň nákladnejšie.

Hlavnou úlohou tejto diplomovej práce je tak navrhnúť bezdrôtový spôsob aktualizácie firmvéru mikrokontroléra (MCU) TI CC1310. Osobným prínosom riešenia takéhoto zadania bolo využitie a rozšírenie znalostí v oblasti

programovania embedded zariadení, ktoré môže vyústiť k reálnemu využitiu v praxi.

Pre splnenie úlohy sa je najskôr potrebné oboznámiť s použitými technológiami. V teoretickej časti tak v prvom rade bude predstavená platforma Internetu vecí spolu s bezdrôtovými technológiami, následne sa v ďalšej kapitole analyzujú možnosti aktualizácie firmvéru na trhu. Na záver teoretickej časti sa popisuje použitá hardvérová platforma spolu s operačným systémom.

Druhá časť práce sa sústredí na návrh praktického riešenia problému a začína demonštráciou všeobecného konceptu návrhu. Neskôr sa prechádza na konkrétnu aplikáciu zo strany klienta (modulu s TI CC1310) a servera (zariadenia riadiaceho aktualizáciu) s ukážkami kódu v jazyku C. Na záver je výsledné riešenie otestované a je zhodnotená jeho funkčnosť spolu s jeho prínosom.

Časť I

Teoretický úvod

Kapitola 2

Internet vecí - IoT

Pojem IoT možno zjednodušene chápať ako rozšírenie pripojenia rôznych zabudovaných elektronických zariadení medzi sebou a do siete Internet. Obvykle je toto spojenie bezdrôtové. Takýto typ pripojenia pritom prináša nové možnosti vrámci interakcie zariadení, ako je napríklad sústredenie dát, ich vyhodnocovanie a následné spracovanie.

Tento princíp pripojenia môže a postupne aj nachádza svoje uplatnenie pri použití meracích, alebo monitorovacích snímačov v priemysle, v zdravotníctve, v domácnosti, dajú sa použiť na sledovanie priebehu, pohybu, alebo polohy technických prostriedkov či tovarov. Snaha pri využití IoT systémov je vo svojej podstate zameraná na autonómne fungovanie bez nutného zásahu človeka, a tým jej zjednodušenie.

Základnou časťou ekosystému IoT sú tzv. embedded, teda vstavané zariadenia. Takéto elektronické súčiastky možno charakterizovať relatívne obmedzenými výpočtovými schopnosťami v zmysle výkonnosti procesora, ako aj veľkosti pamäte. Funkcionalita, ktorú embedded zariadenia majú, začína od tých najjednoduchších, ako je napríklad meranie teploty, až po také komplexné, ako sú smart hodinky s dotykovým displejom a množstvom senzorov. Tieto zariadenia sú následne poprepájané určitou topológiou siete, ktorou sa dostanú potrebné dáta na želané úložisko, a môžu byť ďalej spracované alebo distribuované do koncových zariadení. Toto všetko a ešte ďalšie aspekty zahrňuje pojem IoT.

Nakoľko embedded zariadenia sú súčasťou každodenného života a obsiahnuté v každej elektronile, dalo by sa povedať, že ich pripojenie na sieť Internet vrámci IoT je výustením prirodzenej evolúcie. Podľa spoločnosti Cisco bolo

na prelome rokov 2008 a 2009 takýchto zariadení na svete pripojených už niekoľko miliárd. Pojem sa začal rozširovať a o niekoľko rokov neskôr sa už IoT začalo považovať za štandard, rozrastajúci sa každým rokom. [5] V súčasnosti je počet pripojených zariadení viac ako 6 miliárd a Cisco počíta s tým, že bude každoročne rásť. Podľa odhadov tu bude v roku 2020 asi 40 miliárd pripojených zariadení.

Obrázok 2.1: IoT systémy. Prevzaté z [1]

Návrh a využívanie systému na báze IoT so sebou prináša mnohé úskalia. V počiatočných fázach ide najmä o výber zariadení, snaha o čo najmenšiu spotrebu kvôli výdrži batérie, návrh štruktúry komunikácie a mnoho iného. Povaha prenášaných dát ale môže znamenať hlavne bezpečnostné riziko. Takéto zariadenia je možné využiť okrem domáceho prostredia aj v komerčnom prostredí ako v hospodárstve, lekárstve alebo v doprave. Preto sa dôraz pri výrobe týchto zariadení okrem spoľahlivosti a cenovej dostupnosti kladie aj na bezpečnosť.

Hrozbou súčasnosti je, že sa zariadenia IoT môžu využívať napríklad na kradnutie osobných údajov. Získanie citlivých údajov o používateľovi môže neskôr slúžiť ako podklad za účelom neskoršieho vydierania užívateľov platformy.

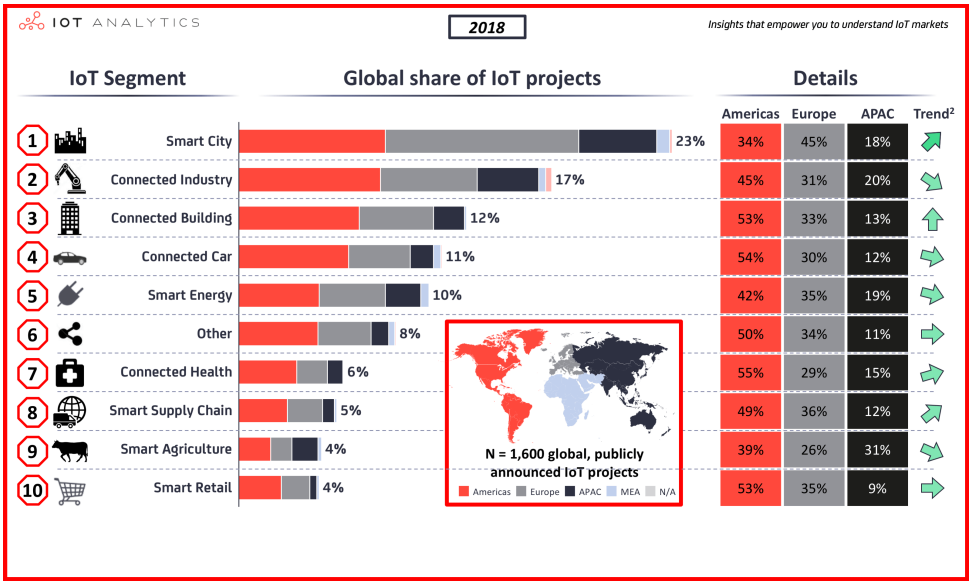
Ďalšou hrozbou IoT zariadení je možnosť byť terčom tzv. DoS alebo DDoS útokov. DoS označuje anglický pojem Denial of Service a znamená znefunkčnenie danej služby. Takéto útoky sú známe zo súčasného internetu. S príchodom IoT zariadení sa výrazne zväčšuje potenciál takýchto útokov, ktoré môžu byť smerované hlavne na veľké energetické, dopravné, alebo mestské infraštruktúry, ktoré stále viac využívajú IoT zariadenia.

Nedostatky v komunikácii potom môže útočník využiť na odcudzenie

identity, osobných údajov, alebo aj využitie výpočtového procesu zariadenia vo svoj prospech. Inteligentne domáce spotrebiče a ich výpočtový proces môže útočník využiť aj na uskutočnenie DoS útoku. Nebezpečné je napríklad, že moderné automobily je možné na diaľku uzamknúť, pripadne zabrzdiť, a keďže inteligentných zariadení neustále pribúda, je tu predpoklad, že sa bude zvyšovať aj riziko zneužitia ich zabezpečenia.

2.1 Využitie IoT

V kombinácii s nedávnymi progresom sú mikrokontroléry jednoduché elektronické zariadenia s nízkou spotrebou, ľahko a lacno prepojiteľné cez internet. Miesto interakcie s človekom používajú tieto systémy snímače a ďalšie pokročilé detekčné mechanizmy. Senzory slúžia na zber údajov. Dáta majú svoju hodnotu a v koncepte IoT hrajú primárnu rolu - sú rozosielané ďalej po sieti, ďalej spracovávané a sú neoddeliteľnou súčasťou systému. Dôležité teda je, v akej podobe budú nakoniec zhromaždené dáta zúžitkované. Obvykle bývajú často ukladané na cloud, teda na úložisko umiestnené mimo zariadenia senzorovej siete, kde sa s nimi ďalej narába a sú zobrazené a použité v konečnej aplikácii.



Obrázok 2.2: Podiel využitia IoT jednotlivých odvetviach. Prevzaté z [2]

Typickým príkladom využitia sú takzvané smart buildings, ktoré využívajú dáta zbierané IoT senzormi a ich analýzou sa snažia vylepšovať podmienky v budove. Obrázok 2.2 zobrazuje trendy a podiely IoT v jednotlivých odvetviach,

pričom práve smart city a smart buildings zaznamenávajú najvačší medziročný nárast.

IoT s využitím senzorov sa spoločne označuje aj ako Wireless Sensor Networks (WSN). Senzory môžu byť vrámci WSN použité na logistiku alebo údržbu a sledovanie stavu rôznych systémov. Americká spoločnosť American Airlines napríklad využíva sieť senzorov monitorujúcich let a dáta využíva na preventívnu údržbu.[6]

Ďalším príkladom IoT je rádiová identifikácia označovaná skratkou RFID. Jedná sa o automatickú identifikáciu za použitia rádiových elektromagnetických vĺn medzi RFID čítačkou a tzv. tagom, ktorý v sebe uchováva svoj unikátny kód [7]. Tento systém býva používaný pri sledovaní procesu výroby, elektronickom mýte, alebo môže byť súčasťou osobných dokladov.

Spomenuté technológie sú iba zlomkom z obrovskej množiny aplikácií. Progres IoT je vďaka pomerne lacnej cene použitého hardvéru a jeho rýchlemu vývoju veľmi rýchlo sa rozrastajúca platforma a jeho možnosti využitia sú limitované v podstate len ľudskou fantáziou.

2.2 Bezdrôtové technológie využívané v IoT

Integrálnou časťou IoT platformy je využitie bezdrôtovej komunikácie. Z fyzikálneho hľadiska je takýto prenos dát založený na prevode informácie do elektromagnetického vlnenia. Pre tento prenos sa zvyčajne používajú pásma Industrial Scientific and Medical, ďalej iba ISM. Je to skupinu frekvenčných pásiem štandardizovaná pre rádiové vysielanie, využiteľná pre vedecké, zdravotnícke a priemyselné prostredie.

K rozdeleniu jednotných pásiem a vyhradeniu pásma ISM došlo historicky práve z dôvodu, že nastala potreba usmerniť využitie a najmä jeho komerčné využitie a obmedziť ho. Metód prenosu dát na malé vzdialenosti je niekoľko. Medzi najvýznamnejšie technológie umiestnené do ISM pásma patria štandardne WiFi, Bluetooth, ZigBee a ďalšie. Pre obojstrannú komunikáciu zariadení je možné implementovať aj proprietárne protokoly. Pomocou nich prebieha komunikácia medzi Internetom vecí a užívateľom.

Pásma sú povolené a používané v homologovaných zariadeniach bez licencií, čo je ich veľkou výhodou. Nedostatkom je, že sú bez garancie proti rušeniu. Všeobecnými licenciami vydanými telekomunikačným úradom ľubovoľného

štátu sú dané aj podmienky záväzné pre používanie týchto zariadení, ako je modulácia, alebo výkon. Napríklad Bluetooth pásmo sa nachádza na frekvencii 2,4 GHz a pásmo IEEE 802.11 označované aj Wi-Fi na frekvenciách 2,4 GHz a 5 GHz [8]. Viac priblížené sú v nasledujúcich sekciách.

2.2.1 Bluetooth

Bluetooth je bezdrôtová technológia umožňujúca dátovú komunikáciu medzi digitálnymi zariadeniami aktuálne na vzdialenosť maximálne niekoľko desiatok metrov. Za jeho vznik môže spoločnosť Ericsson Mobile, ktorá potom neskôr založila s ďalšími spoločnosťami Bluetooth Special Interest Group starajúcu sa o štandardizáciu technológie. Hlavnou funkciou technológie Bluetooth je prepojenie a komunikácia dvoch, alebo viac zariadení na krátku vzdialenosť v domácnosti, alebo v priemyselnom prostredí.

Od Bluetooth 4.0 existuje aj verzia Bluetooth Low Energy (BLE), ktorá je už podľa svojho názvu charakteristická svojou nízkou spotrebou energie. Aby to dosiahla, implementuje tri rôzne režimy spotreby. Táto verzia otvorila príležitosti v domácnosti, v zdravotníctve a v priemysle pre prístroje, ktoré musia komunikovať s vonkajším svetom. Okrem toho dokáže BLE úplne zašifrovať prenášané informácie pomocou šifrovacieho algoritmu AES128 - Advance encryption Standard s 128 bitovou dĺžkou symetrického kľúča.[9]

Jedným zo základným bezpečnostných aspektov je vzájomné párovanie týchto zariadení. Proces párovania pozostáva z procesu výmeny vzájomných identifikačných údajov, ktorými sa zariadenia overia, a následne je možné vymieňať dáta. Bluetooth je dnes štandardom komunikácie na menšiu vzdialenosť, ktorá je súčasťou skoro každého súčasného prenosného počítača, mobilného telefónu alebo aj bezdrôtových slúchadiel.[10]

2.2.2 WiFi

Siete WiFi - Wireless Fidelity, sú založené na štandardoch IEEE 802.11, spravovaných medzinárodnou organizáciou Institute of Electrical and Electronics Engineers. [8]

Existuje viacero postupne vydávaných špecifikácií označovaných písmenom za číslom štandardu. Každá z nich má rozličné vlastnosti prenosu, a výrobcovia podporujú súčasne niekoľko štandardov. Pre budovanie vnútorných

bezdrôtových sietí v priestoroch, kde je inštalácia rozvodov nepotrebná alebo nerealizovateľná, sa dajú s výhodou použiť práve siete WiFi. Technológia je teda určená pre vnútorné použitie a mobilitu v kancelárskych priestoroch.

Dôležitým faktorom pre bezpečnosť WiFi komunikácie je práve šifrovanie dát. Existujú početné typy šifrovania od pomerne málo bezpečnej šifrovacej metódy Wired Equivalent Privacy (WEP) až po WPA/WPA2 WiFi Protected Access, alebo WPA2 Enterprise pre komunikáciu v IoT zariadení s Access point (AP). Prenášané dáta musia byť šifrované, tým sa minimalizuje riziko získania dát treťou stranou odpočúvajúcou komunikáciu. Počet IoT zariadení rastie a organizácia WiFi Alliance preto predstavuje stále nové štandardy pre komunikáciu, ktorú využívajú IoT zariadenia.

Medzi jednu z novších špecifikácií určených práve pre IoT patrí WiFi HaLow, označovaná ako štandard IEEE 802.11ah. Pracuje v bezlicenčnom frekvenčnom spektre s frekvenciou pod 1 GHz, zásluhou čoho je možný aj prechod rádiových vĺn cez prekážky. Je menej stratová, a síce rýchlosť tejto komunikácie v porovnaní s druhými WiFi štandardmi nie je vysoká, pre IoT senzory postačuje, pretože množstvo prenášaných dát je nízke. Prednosťou nízkej prenosovej rýchlosti je, že senzory prenášajú dáta iba v určitom časovom úseku a súčasne nízkou rýchlosťou, a batérie v IoT senzoroch tak vydržia dlhšie.

2.2.3 Sub-1GHz

Častou využívanou technológiou pre IoT je práve Sub-1GHz. Tá už na základe názvu označuje operovanie v pásmach pod 1 GHz, konkrétne v oblastiachg 769-935 MHz, 315 a 468 MHz. Technológia umožňuje prenášať dáta bez použitia veľkého množstva energie na pomerne veľké vzdialenosti. Distribúcia energie, poľnohospodárstvo, zdravotná starostlivosť, výroba a početné ďalšie odvetvia prijali Sub-1GHz ako spôsob komunikácie aplikácií, ktoré potrebujú odosielať pravidelne malé množstvo údajov.

Vzhľadom na hojné využitie v IoT sektore často výrobcovia priamo podporujú túto technológiu s už navrhnutými základnými ovládačmi a Application Programming Interface (API). To môže značne urýchliť čas vývoja aplikácie, čo môže byť v priemysle kľúčovým faktorom prerazenia výrobku na trhu.[11]

Sub-1GHz je výborná voľba napríklad vo veľkých mestách, nakoľko signály s touto frekvenciou sa v mestskom prostredía šíria lepšie ako signály s

frekvenciou 2,4 GHz. Signál sa totižto lepšie ohýba okolo veľkých štruktúr. Komunikácia Sub-1GHz je preto ideálna pre aplikácie, kde sú uzly snímačov na miestach, ktoré sa ťažko obsluhujú, ako sú meteorologické stanice a inteligentné merače. Sub-1GHz je tiež menej náchylný na rušenie, nakoľko jeho frekvenčné pásmo nie je tak využívané. Vzhľadom na povahu siete využívanej aj v riešenom zadaní a potrebe čo najnižšej spotreby bola táto technológia využitá aj pri aktualizácii aplikačného firmvéru.[12]

Kapitola 3

Analýza spôsobov prevedenia aktualizácie firmvéru embedded procesorov

Spočiatku bola aktualizácia firmvéru embedded zariadení braná ako výnimočná udalosť potrebná za určitých okolností. Takúto situáciu následne riešil kvalifikovaný pracovník. Rýchlejší vývoj elektroniky a sofvéru, ktorý ju riadi, ale vyžadoval zmenu. Je dôležité rýchlo reagovať na rôzne bezpečnostné hrozby a drobné chyby odhalené až počas reálnej prevádzky zariadenia.

Z týchto dôvodov sa proces aktulizácie firmvéru vyvinul v bežnú operáciu vykonávanú zvyčajne automaticky. Neoddeliteľnou súčasťou procesu aktualizácie je tzv. bootloader, tiež označovaný aj ako zavádzač firmvéru. Ide o časť kódu, ktorá zvyčajne ostáva nezmenená (viac v 3.2.2), a rozhoduje o obraze firmvéru, ktorý bude načítaný a spustený. Závádzač má rôzne spôsoby implementácie, o ktorých sa bude hovoriť v nasledujúcich sekciách 3.1 a 3.2. [13]

3.1 Možnosti zabezpečenia procesu aktualizácie firmvéru

Pokiaľ sa vstavané zariadenie dostane do procesu aktualizácie svojho firmvéru, nastáva pre jeho celkovú integritu a funkčnosť veľké bezpečnostné riziko. Pri narušení priebehu prijímania chybou v zavádzači, alebo pri prenose nového obrazu, poprípade vplyvom útoku z tretej strany sa môže zariadenie stať úplne nepoužiteľné, prípadne môže prísť o cenné dáta. Chyby pri spracovaní alebo pri prenose sa riešia kontrolnými mechanizmami vnútri zavádzača, proti

možným útokom z tretích strán je možné použiť ďalšie prostriedky. Jedným z nich je možnosť využitia procesora alebo bootloadera s šifrovaním. V takomto prípade je potrebné bezpečne uschovávať šifrovací kľúč a poznať aktuálny stav šifrovacieho jadra. Pri šifrovaní obrazu sa môže využiť metóda asymetrickej enkrypcie, kedy sa generuje pár privátny kľúč-verejný kľúč. Server vtedy zašifruje obraz so svojím privátnym kľúčom, ktorý má iba on. Bootloader so šifrovaním potom dokáže overiť integritu prichádzajúceho obrazu s verejným kľúčom, či nedošlo k jeho neželanej modifikácii. Takýto bootloader už používa napríklad firma Texas Instruments v rade mikrokontrolérov MSP [14]. Tiež je možné verifikovať podpis bootloadera pri jeho prvom zavádzaní pri štarte pamäťou ROM. Takto sa ROM uistí, či nebol narušený bootloader. Ďalšou možnosťou je využitie vyššieho výpočtového výkonu vstavaných zariadení a softvérovej výbavy na analýzu možných útokov a detekciu nakazených paketov.[15]

3.2 Spôsoby aktualizácie firmvéru

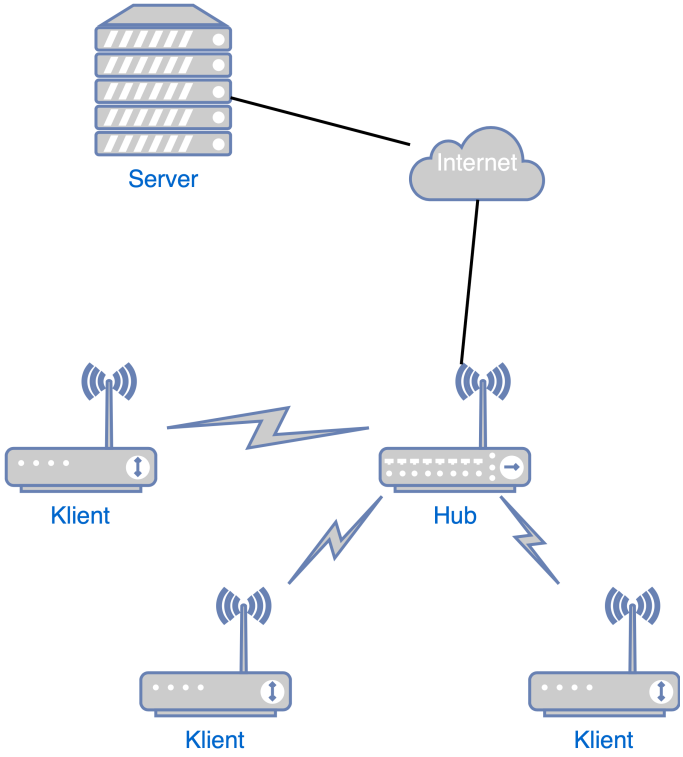
Možnosti aktualizácie firmvéru je možné rozdeliť do dvoch veľkých skupín, a to na klasický drôtový update a bezdrôtový update firmvéru. Oba typy riešenia sa bežne neustále používajú a sú popísané v nasledujúcej časti.

3.2.1 Aktualizácia firmvéru s použitím fyzického pripojenia

Bežnou metódou výmeny firmvéru je využitie programátora, ktorý je uspôsobený na túto funkciu. Ide teda o rýchly a bezpečný proces, keďže programátor aj overuje, či sa firmvér nahral do zariadenia správne. Slabou stránkou takéhoto postupu ale je potreba mať dané zariadenie fyzicky pri sebe na pripojenie k programátoru. Pokiaľ je potrebné aktualizovať viacej zariadení naraz, môže to byť veľmi nepraktické. Čo sa používaných rozhraní týka, za zmienku určite stojí napríklad rozhranie priamo určené na programovanie - JTAG [16], ktoré môže tiež slúžiť na ladenie softvéru a hľadanie chýb. Takýto spôsob sa často používa hlavne pri vývoji firmvéru, kedy je používané zariadenie priamo k dispozícii a ešte nie je v prevádzke. Okrem rozhrania JTAG ale môžu byť využité aj iné technológie, medzi nimi aj štandardné USB, v takom prípade ale už zariadenie obvykle obsahuje vhodne prispôsobený bootloader.

3.2.2 Bezdrôtová aktualizácia firmvéru - OTA

Bezdrôtový spôsob aktualizácie firmvéru je postupne čoraz obľúbenejší a veľký podiel má na tom najmä IoT. Dôležitým faktorom pri výbere tohto riešenia je totiž jeho pohodlnosť, nakoľko nemusí dôjsť k fyzickému prepojeniu so zariadením obsahujúcim nový obraz. V prípade veľkej siete typu mesh s množstvom uzlov rozmiestnených na veľké vzdialenosti dochádza pri využití OTA k nesmiernemu zjednodušeniu procesu, no dopad je cítiť už aj pri menšom množstve zariadení. V prípade OTA zvyčajne riadi celý proces server, ktorý komunikuje s klientami a poskytuje im nový obraz firmvéru. [17]



Obrázok 3.1: Koncept bezdrôtovej aktualizácie

V zásade pri procese prenosu obrazu ide o konverziu nového firmvéru na sekvenciu bajtov, teda klasický binárny formát s príponou *.bin* alebo *.hex*, ktorý obsahuje bajty určené pre presnú adresu v pamäti. Zvyčajne je tento finálny súbor príliš veľký, aby sa mohol odoslať naraz, v tom prípade sa musí súbor rozdeliť na viacero paketov. Pri bezdrôtovej komunikácii v IoT však často ide o čo najväčšiu úsporu energie klienta, preto sa môže využiť zmenšenie obrazu kompresným algoritmom, ktorý sa použije na celý obraz. Pokiaľ sa oproti predošlej verzii obrazu mení len časť kódu, môže sa využiť tzv. inkrementálny update. Vtedy sa prenáša iba rozdiel medzi verziami za

pomoci rozdielových algoritmov, ktorých výstupom je tzv. delta kód. [18][13]

Zásadný podiel na náraste bezdrôtových aktualizácií majú tiež čoraz rozšírenejšie a sofistikovanejšie bezdrôtové protokoly a rozhrania. Pokiaľ je zariadenie pripojené na internetovú sieť, je možné prevádzkovať OTA aj s jeho využitím. Okrem bezdrôtového pripojenia na internetovú sieť je ale možné využiť aj množstvo periférií ako je ZigBee, WiFi, Sub-1GHz, Bluetooth a iné.[18]

Práve posledná spomínaná technológia Bluetooth býva už natívne podporovaná niektorými výrobcami (napr. ST [19], Cypress [20]) pre bezdrôtovú aktualizáciu firmvéru. Jej využitie je pritom logické, ide o jednu z najrozšírenejších bezdrôtových technológií dnes implementovanú v každom laptope a mobilnom telefóne. Okrem toho od verzie 4.0 existuje verzia tzv. BLE a tá od verzie 4.2 obsahuje bezpečnostné mechanizmy, ktoré chránia pred útokmi tretích strán a odpočúvaniu komunikácie.

V prípade OTA existujú pre spôsob implementácia bootloadera dve základné možnosti. Prvou je fixný bootloader, v tomto prípade sa jeho kód nikdy nemení, mení sa iba firmvér procesom aktualizácie. Pokiaľ je bootloader fixný, nachádza sa v sekcii pamäte chránenej proti zápisu. Jeho základnou výhodou je rýchlejší čas aktualizácie, ak sa však v takomto bootloaderi nájde akákoľvek chyba, jeho kód nemôže byť upravený. Takúto situáciu rieši tzv. aktualizovateľný bootloader.[17] Každý výrobca má svoj vlastný spôsob implementácie, ale vo všeobecnosti sa pamäť dá v takom prípade rozdeliť na tri základné časti:

Launcher v podstate bootloader bez komunikačného rozhrania. Vykonáva kontrolu, či má dôjsť k aktualizácii, a na základe toho sa beh posunie do časti Stack alebo Application. Tiež aktualizuje časť Stack, pokiaľ je k dispozícii nový obraz.

Stack v sebe uchováva BLE stack, teda súhrn protokolov slúžiacich na obsluhu BLE. Hlavnou funkciou tejto časti je komunikácia so zariadením, ktorý má k dispozícii nový obraz firmvéru. Jej vlastný obraz nahráva Launcher a Stack sa následne stará o aplikačnú časť. Je to teda časť booloadera, ktorú je možné aktualizovať.

Application Kód aplikácie, o zavádzanie nového obrazu sa stará Stack.

Na zavádzače pre OTA vstavaných zariadení sa dá ešte pozerať z pohľadu využitia pamäte. Nový obraz stacku alebo aplikačného firmvéru môže byť

napríklad zapísaný priamo do vnútornej Flash pamäte. To vylučuje potrebu externej pamäte a samotná implementácia je tak jednoduchšia. V druhom prípade môžu byť nové obrazy uložené na externú pamäť. Následne sa aktualizuje interná Flash pamäť, preto sa musí nový obraz prečítať cez komunikačné rozhranie externej pamäte. Táto možnosť sa využíva, ak má zariadenie nedostatočnú vnútornú kapacitu pamäte.[17]

Kapitola 4

Použitá hardvérova platforma a operačný systém

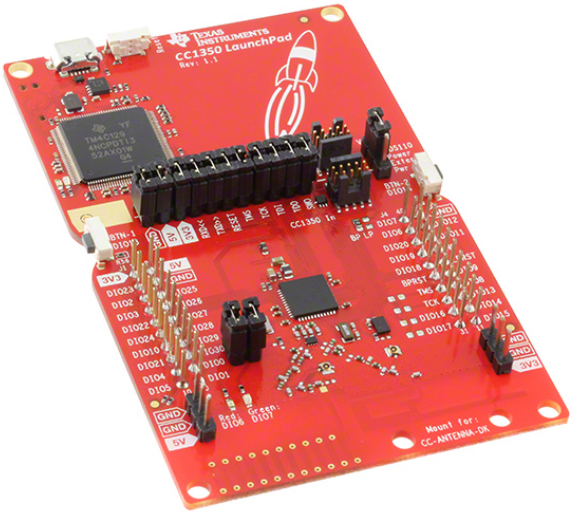
Pred samotným popisom návrhu bude predstavená použitá hardvérová platforma spolu so softvérovou vrstvou, aby bol získaný celkový prehľad o možnom riešení.

4.1 TI CC1310

Produkt CC1310 od výrobcu Texas Instruments je zadanou platformou pre vývoj aktualizácie firmvéru. Mikrokontrolér patrí do rodiny s názvom SimpleLink, ktorá zahrňuje vybrané MCU so širokou podporou týchto drôtových rozhraní, ako je Ethernet, zbernica CAN a USB, ale aj bezdrôtových periférií. Do tejto skupiny patria technológie Bluetooth, Sub-1GHz, ZigBee, Thread a Multi-standard. Výhodou tejto platformy je jednotná knižnica pre spomínané technológie, vyvinutý softvér pre jeden mikrokontrolér z tejto skupiny je teda možné použiť aj pre ostatné bez akýchkoľvek zmien. SimpleLink riešenia tiež natívne podporujú kryptografické štandard ako napríklad AES-128, ale aj sieťové šifrovania, konkrétne WPA2 alebo TLS. Tieto všetky vlastnosti sú podporované pri veľmi nízkej spotrebe.[21]

4.1.1 Procesor Arm Cortex-M3

Použitý mikrokontrolér obsahuje jadro 32 bitového procesora Arm Cortex-M3. Tento procesor je založený na architektúre Armv7 od firmy Arm Holdings, ktorá je pokračovateľom tradície Arm procesorov a terajším štandardom v



Obrázok 4.1: CC1310 Launchpad [3]

mobilných zariadeniach. Názov Arm pochádza zo skratky Advanced RISC Machine, odvodený z RISC inštrukčnej sady, ktorú prvé Arm procesory využívali. RISC architektúry sa vyznačujú redukciou počtu inštrukcií, ich zjednodušením a zároveň zjednodušením dátových ciest a riadiacej jednotky. Zložité výpočty sa teda v RISC vykonávajú ako postupnosť viacerých jednoduchších inštrukcií.[22]

Využitá architektúra Armv7-M má druhotné označenie M podobne ako v názve procesoru, ktoré indikuje profil použitia, na aké sa procesor špecializuje. V tomto prípade M znamená mikrokontrolér a procesor je teda určený pre aplikácie zamerané na energetickú účinnost, nízku spotrebu a malú plochu čipu. Klasickým príkladom je teda použitie v IoT sektore. Tieto procesory obsahujú iba inštrukčnú sadu T32 označovanú aj Thumb-2. T32 kombinuje inštrukcie 16 a 32 bitovej dĺžky, čo môže kompilér využiť na vyváženie veľkosti kódu. Na zvýšenie rýchlosti vykonávania inštrukcií využíva trojstupňové reťazové spracovávanie (pipelining). [23]

Okrem podpory jedinej inštrukčnej sady je procesor Cortex-M3 odlišný aj ďalšími spôsobmi prevedenia architektúry. Neobsahuje tzv. Current Program Status Register (skrátene CPSR), ktorý uchováva informácie o aktuálnom statuse procesora. Ďalej automaticky ukladá a obnovuje stav v prípade

výnimiek. Cortex-M3 tiež integruje ovládač prerušení.[24]

4.1.2 Pamäte

Microcontroller Unit (MCU) disponuje programovateľnou 128 KB Flash pamäťou, ktorá je rozdelená na 4 KB bloky. Jednotlivé bity sú predvolene v stave logickej 1 a môžu byť naprogramované samostatne na hodnotu logickej 0. Vymazať tento stav je možné iba pre celý blok naraz. Okrem Flash pamäte má CC1310 k dispozícii 20 KB SRAM a ROM pamäť, ktorá slúži na uloženie operačného systému a bootloaderu.

4.1.3 RF jadro

Zariadenie CC1310 obsahuje nízkopríkonový RF vysielač a prijímač, ktorý je obsluhovaný samostatným procesorovým jadrom Cortex-M0. To sa stará o protokolové príkazy na nízkej úrovni a tak zabezpečuje úsporu výkonu hlavného procesora a väčšiu flexibilitu.

4.1.4 Riadenie spotreby

Mikrokontrolér CC1310 pre minimalizáciu spotreby podporuje množstvo šetriacich módov a prvkov. Klasický beh povolených periférií a procesora vykonávajúceho kód je označený ako *active*. Stav *idle* povoľuje hodinový signál do zapnutých periférií, ale nie do procesora, ktorý tak nevykonáva svoju funkciu. Ten sa môže vrátiť do módu *active* pri akejkoľvek udalosti prerušenia. Podobným režimom je *standby*, na vrátenie do režimu *active* je ale potrebná udalosť z tzv. Always-On (AON) domény. Do AON domény patrí externá udalosť, udalosť z Real-time čítača (RTC) a udalosť z ovládača sensorov. Pod pojmom ovládač senzorov sa označuje autonómny procesor, ktorý ovláda periférie nezávisle na hlavnom procesorovom jadre. Posledný režim je *shutdown* režim, v ktorom je zariadenie kompletne vypnuté, a môže byť prebudené zmenou na akomkoľvek vstupno-výstupnom pine, ktorý je označený ako *wake from shutdown pin*. Vzniknutá udalosť tak slúži ako spúšťač resetu.[21]

4.2 Operačný systém

V nasledujúcej časti bude popísaný použitý operačný systém a možnosti, ktoré boli na výber. Pre uvedenie do problematiky operačných systémov je tiež naznačené delenie operačných systémov a dôvody ich použitia pre rôzne účely.

Staršie modely MCU boli z veľkej časti navrhnuté bez operačného systému. Zvyčajne mala implementácia požadovanej funkcie podobu programu, ktorý vykonával kód v nekonečnej slučke. Vzhľadom na rapídne sa zvyšujúcu hustotu integrácie a klesajúcu cenu sa však rozdiely vo výkone a možnosti medzi mikrokontrolérmi a procesormi riadiacimi osobné počítače postupne vymazávajú. Čím viac úloh sa teda od mikrokontroléra požaduje vykonávať, o to viac je použitie operačného systému potrebné na ich správu a organizáciu.

Úlohou operačného systému je vo všeobecnosti správa hardvérových prostriedkov a ich prideľovanie pre bežiace aplikácie, ktoré si ich žiadajú. Zväčša preto obsahujú nevyhnutné komponenty ako správa pamäte, súborové systémy, multitasking a plánovač udalostí. Existujú rôzne typy operačných systémov. Osobné počítače (z anglického výrazu "personal computer") využívajú všeobecne zamerané operačné systémy (ako napríklad Microsoft Windows alebo distribúcie Linuxu), ktoré sú orientované na obsluhu okien používateľov. Tieto operačné systémy sa teda nesústreďujú na precízne časovanie úkonov, ale na čo najlepšiu užívateľskú odozvu.

4.2.1 RTOS

RTOS je skratka označujúca Real Time Operating System, teda systém operujúci v reálnom čase. Tento typ operačného systému je špeciálne navrhnutý na vysokú spoľahlivosť a presné časové parametre úloh. Pritom neimplementuje klasický multitasking, teda paralelné spracovanie viacerých vlákien naraz, ale veľmi rýchlo prepína medzi jednotlivými vláknami.

Aby vedel RTOS deterministicky určiť poradie a dĺžku trvania vlákien, je im priradená určitá priorita a je potrebné vedieť odhadnúť čas, v ktorom sa bude vlákno spracovávať s maximálnou možnou chybou.

V RTOS sa na základe časových odhadov dajú systémy rozdeliť na tzv. hard real-time a soft real-time. Hard real-time operačné systémy vedia s

absolútnou istotu garantovať maximálnu dobu trvania jednej úlohy, zatiaľ čo soft real-time to dokážu iba vo väčšine prípadov. Výber medzi týmito dvomi variantmi závisí od potreby presnosti časových odhadov. Rôzna dĺžka trvania jednej úlohy pre jednotlivé iterácie sa tiež popisuje hodnotou parametru jitter. Úlohou RTOS je teda minimalizácia tejto hodnoty.[25]

S ohľadom na špecifiká RTOS je jeho využitie časté najmä v oblasti automatizácie a meraní, pre ktoré je presné časovanie kritické. Jeho väčšia komplexnosť oproti jednoduchšiemu klasickému prístupu programovania MCU ho tiež predurčuje na riadenie väčšieho množstva procesov spolu s externými zdrojmi podnetov. Takýmto prípadom použitia RTOS je aj riešené zadanie tejto práce, ktoré sa stará najmä o zber a následné zdieľanie dát po vytvorenej sieti. [26]

4.2.2 TI-RTOS

Existuje množstvo ponúkaných RTOS priamo od výrobcov procesorov, poprípade vyvinutých treťou stranou. Príkladom RTOS od tretej strany je aj najpoužívanejší RTOS licentovaný univerzitou MIT - The FreeRTOS Kernel. Tento open source projekt je možné implementovať pre množstvo mikrokontrolérov od rôznych výrobcov. Dopredu vytvorený a odladený RTOS dokáže markantne ušetriť čas potrebný na návrh základných funkcií systému.[27]

Procesor CC1310 má v aktuálnom riešení implementovaný operačný systém TI-RTOS, ktorý je vytvorený priamo spoločnosťou Texas Instruments špeciálne pre ich embedded procesory. Jeho súčasťou je real-time kernel s ďalšími komponentami ako ovládače periférií so svojimi API, súborový systém FAT a podpora USB či TCP/IP protokolov.

Kernel je pomenovaný SYS/BIOS a je navrhnutý s ohľadom na minimalizáciu požiadaviek na pamäť a procesor, a stará sa o real-time plánovanie úloh a konfiguráciu nástrojov. TI-RTOS tiež plne využíva výhody nízko príkonových procesorov. Napájací manažér ovláda distribúciu hodinového signálu,napájanie jednotlivých blokov, a implementuje agresívny stand-by mód, ktorý minimalizuje spotrebu.[28]

TI-RTOS bol ako platforma pre vývoj zvolený okrem vyššie spomínaných vlastností aj z ďalších dôvodov. Jednými z nich sú široké možnosti technickej podpory na fórach firmy Texas Instruments, detailná dokumentácia a previazané kvalitné vývojové prostredie. Pri rozhodovaní zavážili aj skúsenosti a

predošlé použitie operačného systému v rámci projektu.

4.3 Softvérové nástroje

Hlavným nástrojom pre vývoj bezdrôtového upgradu firmvéru bol Code Composer Studio od firmy Texas Instruments. Ide o integrované vývojové prostredie (IDE) určené na kompletný vývoj firmvéru pre ich rodiny mikrokontrolérov a pozostáva zo vstavaného C/C++ kompiléra, editora zdrojových kódov, debuggera a ďalších komponentov, ktoré vedú vývojára krok po kroku až po finálne riešenie projektu.[29]

4.3.1 SimpleLink SDK

Softvérový vývojový kit SimpleLink je navrhnutý pre možnosť vývoja softvéru v jednom spoločnom prostredí. Pre dosiahnutie takéhoto riešenia v sebe implementuje štandardizované ovládače všetkých periférií a k nim príslušný middleware - teda medzivrstvu prepájajúca ovládače s užívateľskou aplikáciou. Okrem spomenutých súčastí je Software Development Kit (SDK) rozšíriteľný o rôzne pluginy, napríklad pre senzory, cloud alebo IoT. Ako bolo spomenuté v Sekcii 4.1, návrh aplikácie za použitia SimpleLink SDK vedie k úplnej prenositeľnosti kódu v rámci mikrokontrolérov rodiny SimpleLink.[30]

Nakoľko bude v práci využitá technológia Sub-1GHz, je potrebné spomenúť aj middleware EasyLink. Ide o vrstvu na vrchu CC1310 RF ovládača, určená pre vytváranie proprietárneho protokolu a je zahrnutá v SimpleLink SDK.

Časť II

Realizácia

Kapitola 5

Koncept riešenia

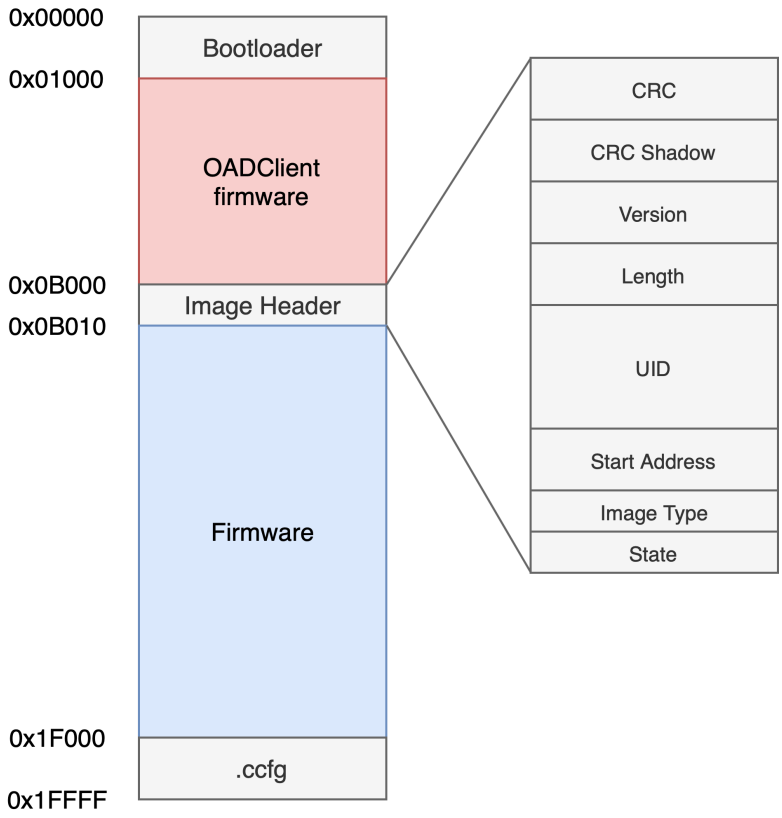
Prvým krokom praktického návrhu je rozvrhnutie jednotlivých úkonov a funkcií. Je potrebné si preto najskôr predstaviť základné črty bootloadera a komunikačný protokol, ktorý bude v celej práci použitý.

5.1 Bootloader

Bootloader je dôležitou súčasťou procesu aktualizácie, ktorého implementáciu je potrebné si rozmyslieť. Ako bolo už popísané v Kapitole 3, jeho primárnou úlohou je zabezpečiť spustenie vlastného firmvéru. Bootloader preto musí mať informáciu o pozícii jednotlivých firmvéroch nachádzajúcich sa v pamäti Flash, musí vedieť vyhodnotiť ich integritu a správne vybrať obraz, ktorý má spustiť. Pre zabezpečenie správneho fungovania procesu bezdrôtovej aktualizácie firmvéru je potrebné, aby Flash pamäť obsahovala aspoň dva od seba nezávislé obrazy firmvéru. Aby bootloader mohol správne určiť spustený firmvér, musia sa tieto sa tieto obrazy nachádzať na presne určených miestach a obsahovať základné informácie pre bootloader.

5.1.1 Rozdelenie pamäte

Programovateľná Flash pamäť mikrokontroléra CC1310 je rozdelená na 32 blokov o veľkosti 4096 bajtov, preto je žiadúce, aby jednotlivé obrazy programov rešpektovali tieto rozdelenia. Bootloader ako prvý proces po spustení sa nachádza na adrese 0x0000. Jednotlivé bloky sú zobrazené na obrázku 5.1.



Obrázok 5.1: Rozdelenie internej flash pamäte

5.1.2 Obraz Firmvéru

Každý obraz firmvéru uložený v pamäti Flash musí obsahovať základné informácie o obraze samotnom. Tieto informácie sa nachádzajú v štandardnej OAD hlavičke na úvode obrazu firmvéru o veľkosti 16 B. Obsahujú potrebné informácie pre bootloader, aby vedel vyhodnotiť integritu obrazu, jeho verziu a schopnosť spustenia sa.

Cyclic redundancy check

CRC je jednoduchá funkcia používaná na kontrolný súčet hlavne pri bezdrôtových prenosoch. Je obľúbená hlavne kvôli svojej jednoduchosti a vysokej spoľahlivosti. Používa sa na overenie integrity obrazu firmvéru, ktoré prebieha v dvoch krokoch. Ako prvé je potrebné vypočítať CRC pri generovaní obrazu, táto hodnota sa uloží do hlavičky a je prenesená zároveň s celým obrazom na koncové zariadenie. Neskôr bootloader overí integritu dát spočítaním CRC

Pole	Veľkosť	Popis
CRC	2	Cyclic Redundancy Check
CRC Shadow	2	Overenie CRC
Version	2	Verzia
Length	2	Veľkosť obrazu
UID	4	Identifikácia
Start Address	2	Adresa v pamäti
Image Type	1	Typ obrazu
State	1	Status

Tabuľka 5.1: Popis hlavičky OAD protokolu.[4]

dostupného obrazu a porovnaním s hodnotou v hlavičke. Pre výpočet je použitý konkrétne algoritmus CRC-16-CCITT, ktorý je široko využívaný napríklad pri prenosoch Bluetooth.

Verzia

Verzia obrazu sa využíva k overeniu aktuálnosti obrazu firmvéru. V prípade, že je dostupný nový obraz, je zahájený jeho prenos.

Dĺžka

Údaj o celkovej dĺžke obrazu v slovách, veľkosť jedného slova je 4 bity.

Status bajt

Posledný bajt hlavičky je využitý na uloženie statusu firmvéru. Slúži na overenie schopnosti firmvéru vykonávať požadované funkcie. Pri generovaní obrazu je status bajt nastavený na hodnotu 0xFF, čo pre bootloader znamená, že obraz je nový a ešte nebol spustený. Ak bootloader vyhodnotí, že obraz si po prenose zachoval integritu, nastaví MSB statusu na hodnotu nula (ukážka 5.1) a pokračuje v jeho spustení.

Hodnota	Popis
0xFF	Nový obraz firmvéru
0x7F	Chýbajúce potvrdenie
0x3F	Plne funkčný obraz firmvéru
0x0F	Je dostupná aktualizácia

Tabuľka 5.2: Hodnoty poľa State v hlavičke OAD protokolu

```
/* Overenie status bajtu v hlavičke obrazu firmvéru */
bool loadImg(void)
{
    uint8_t imageHeaderBuf[16];

    /* Prečítanie hlavičky o veľkosti 16 B
     * Obraz sa nachádza v 10 bloku flash
     * 1 blok má 4096 B
     */
    memcpy(imageHeaderBuf, (uint32_t *)((10 * 4096)), 16);

    /* Obraz firmvéru je nový je povolené spustenie */
    if(imageHeaderBuf[15] == 0xFF) {
        /* Označenie obrazu ako spustený obraz bez potvrdenia */
        Bim_intWriteStatus(((10 * 4096) + 15), (uint8_t *)0x7F);
        return true;
    /* Obraz firmvéru je overený, je povolené spustenie */
    } else if(imageHeaderBuf[15] == 0x3F) {
        return true;
    }
    /* V opačnom prípade je potrebné spustiť firmvér OADClient */
    return false;
}
```

Ukážka 5.1: Overenie status bajtu OAD hlavičky

Pred spustením samotného firmvéru bootloader nastaví interný Watchdog mikrokontroléra. Firmvér po spustení musí tento watchdog resetovať a nastaviť druhý MSB na hodnotu nula, čím informuje bootloader, že funguje správne. V opačnom prípade Watchdog resetuje mikrokontrolér.

5.1.3 Spustenie firmvéru

Spustenie samotného firmvéru zabezpečuje jednoduchá sekvencia strojového kódu. Ako prvé je potrebné nastaviť adresu zvoleného obrazu v pamäti, pre prvý obraz zabezpečujúci celý proces bezdrôtovej aktualizácie je to adresa

0x1010 a pre samotný program je to 0xB010.

```
MOV R0, #0x1010
LDR R1, [R0, #0x4]
```

Stack pointer je malý register, ktorý ukladá informácie týkajúce sa návratovej adresy pri prerušeniach programu, a pred spustením nového obrazu je potrebné ho resetovať.

```
LDR SP, [R0, #0x0]
```

Nasleduje skok na pripravenú adresu v registri R1.

```
BX R1
```

5.2 Prenos obrazu firmvéru

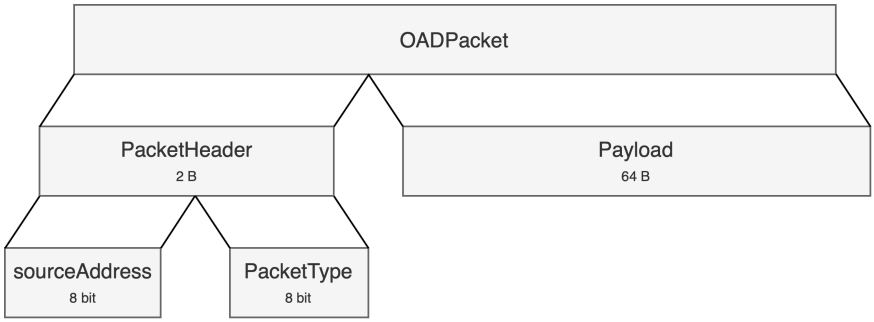
Medzi hlavné úlohy pri bezdrôtovej aktualizácii firmvéru patrí bezpečný prenos obrazu firmvéru zo servera na cieľové zariadenie, respektíve klienta. Tento prenos prebieha na Sub-1Ghz sieti, pričom riadenie RF jadra mikrokontroléra zabezpečuje vrstva EasyLink.

5.2.1 EasyLink vrstva

Vrámci SimpleLink SDK vrstva EasyLink poskytuje určitú abstrakciu riadenia rádiového prenosu pre klientskú aplikáciu. Pomocou malého množstva funkcií je možné jednoducho nastaviť potrebné parametre rádiového prenosu a riadiť komunikáciu.

RF jadro je inicializované funkciou *EasyLink_Init()*, ktorá nakonfiguruje požadované nastavenia ako sú frekvencia, intenzita signálu, rýchlosť prenosu či veľkosť správ. Túto konfiguráciu je možné neskôr upravovať pomocou funkcií *EasyLink_SetFreq()*, *EasyLink_SetRfPw()*, ktoré upravia frekvenciu, respektíve silu vysielaného signálu bez nutnosti znovu konfigurovať celé RF jadro.

Príjem správ je inicializovaný funkciami *EasyLink_receive()* alebo *EasyLink_receiveAsync()*, a môže byť spustený okamžite alebo naplánovaný na konkrétny čas. *EasyLink_receiveAsync()* narozdiel od druhej spomenutej



Obrázok 5.2: OADPacket

funkcie pracuje asynchrónne a vďaka tomu mikrokontrolér môže vykonávať inú činnosť, kým čaká na komunikáciu. API však neumožňuje reťazenie správ, preto pred jeho volaním je potrebné ukončiť asynchrónne prijímanie komunikácie volaním *EasyLink_abort()*. V prípade odosielania správ sa používajú funkcie *EasyLink_transmit()* a *EasyLink_transmitAsync()* s obdobnými vlastnosťami.

5.2.2 OAD Protokol

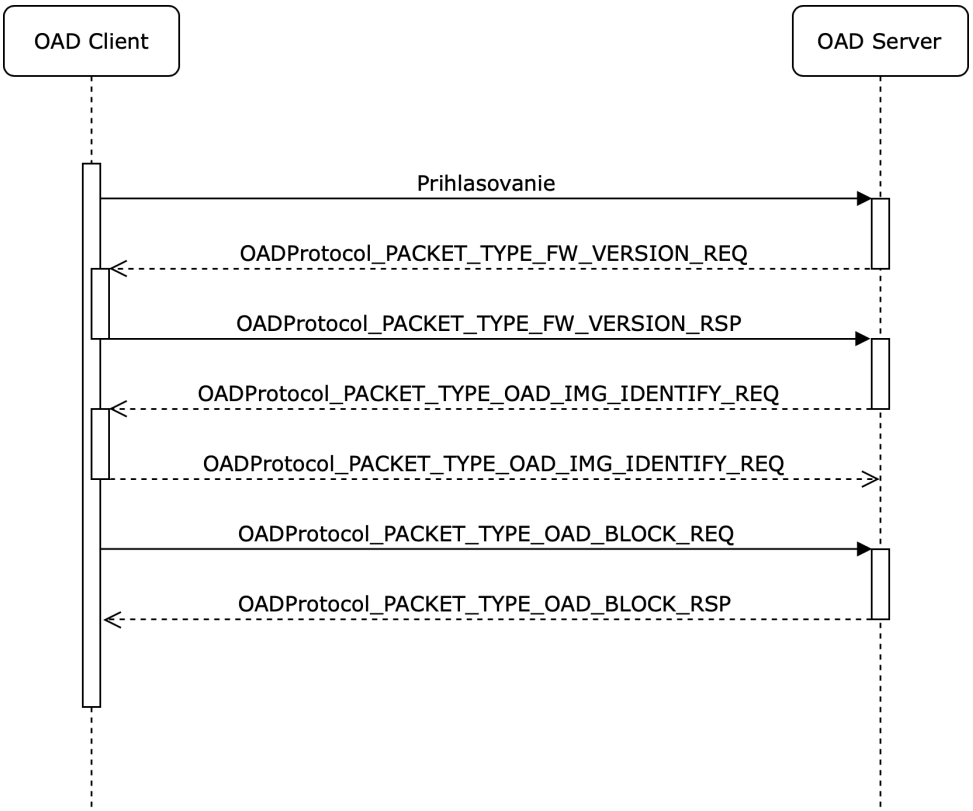
Samotný prenos obrazu nového firmvéru je založený na základe protokolu Over the Air Download (OAD) spoločnosti Texas Instruments. Protokol OAD bol vyvinutý práve kvôli možnosti aktualizovať firmvér mikrokontroléra a daný mikrokontrolér ho musí podporovať. Jedná sa o softvérovú nadstavbu vrstvy EasyLink. Základom protokolu je štruktúra paketu OadPacket, ktorá predstavuje payload každej správy, kde PacketHeader obsahuje identifikáciu odosielateľa a typ odosielanej správy. OadPayload obsahuje samotnú správu, respektíve odosielané dáta o veľkosti 64 bajtov. Štruktúra je naznačená na obrázku 5.2.

```
1 struct PacketHeader {
2     uint8_t sourceAddress;
3     uint8_t packetType;
4 };
5
6 struct OadPacket {
7     struct PacketHeader header;
8     uint8_t oadPayload[OAD_BLOCK_SIZE + 2 + 2];
9 };
```

Ukážka 5.2: Štruktúra OADProtokol

Hodnota	Názov
0x01	PACKET_TYPE_FW_VERSION_REQ
0x02	PACKET_TYPE_FW_VERSION_RSP
0x03	PACKET_TYPE_OAD_IMG_IDENTIFY_REQ
0x04	PACKET_TYPE_OAD_IMG_IDENTIFY_RSP
0x05	PACKET_TYPE_OAD_BLOCK_REQ
0x06	PACKET_TYPE_OAD_BLOCK_RSP

Tabuľka 5.3: Hodnoty poľa packetType v štruktúre PacketHeader



Obrázok 5.3: Komunikácia medzi serverovou a klientskou aplikáciou

OADProtocol_PACKET_TYPE_FW_VERSION_REQ

Správu odosiela vždy server klientskej aplikácii a ako odpoveď požaduje informáciu o aktuálnom firmvéri. Správa nemá žiadny obsah.

OADProtocol_PACKET_TYPE_FW_VERSION_RSP

Odpoveď na *OADProtocol_PACKET_TYPE_FW_VERSION_REQ*, dĺžka správy je 2 bajty a ich hodnota vyjadruje verziu aktuálneho firmvéru vo Flash pamäti klienta.

OADProtocol_PACKET_TYPE_OAD_IMG_IDENTIFY_REQ

Správa obsahuje hlavičku obrazu firmvéru, ako sú informácie o verzii, počte blokov a CRC hodnote dostupného obrazu na serveri. Dĺžka správy je 16 bajtov.

OADProtocol_PACKET_TYPE_OAD_IMG_IDENTIFY_RSP

Potvrdenie klientskej aplikácie, že je schopná prijať nový obraz firmvéru. Dĺžka správy je jeden bajt.

OADProtocol_PACKET_TYPE_OAD_BLOCK_REQ

Správou *OADProtocol_PACKET_TYPE_OAD_BLOCK_REQ* sa začína samotný prenos blokov nového obrazu. Dĺžka správy je 2 bajty a obsahuje poradové číslo požadovaného bloku.

OADProtocol_PACKET_TYPE_OAD_BLOCK_RSP

Samotná správa s blokom obrazu firmvéru o veľkosti 64 bajtov s potvrdením o poradí bloku o dĺžke 2 bajty.

Kapitola 6

Aplikačná vrstva

Samotné aplikácie pre klienta a server stavajú na protokole popísanom v Sekcii 5.2. Ide teda o dve časti vo vyššej vrstve návrhu, ktoré medzi sebou komunikujú pri aktualizácii obrazu firmvéru. Táto práca sa teda nezaoberá cieľovým firmvérom pre koncové zariadenia a ani ich komunikáciou so serverom, nakoľko je úplne nezávislá od finálnej aplikácie.

6.1 OADClient

Samotný proces stiahnutia a uloženie nového obrazu firmvéru riadi aplikácia OADClient. OADClient je spolu s bootloaderom nahraná do pamäte Flash pri výrobe zariadenia. Napriek tomu, že sa nachádza v užívateľsky prístupnej časti pamäte, nie je prispôsobená na vlastný update z dôvodu šetrenia veľkosti pamäte pre samotnú klientskú aplikáciu. Spustenie obrazu OADClient môže mať niekoľko dôvodov. Ako prvý je prirodzene prvé spustenie zariadenia, kedy sa na zariadení nachádza jediný obraz firmvéru. Ďalšími možnosťami sú iniciovanie aktualizácie klientskou aplikáciou alebo neschopnosť vykonávať požadované funkcie klientskou aplikáciou, pričom v oboch prípadoch musí byť dostupná aktualizácia.

Po spustení obrazu OADClienta a inicializácii modulov s operačným systémom TI-RTOS je skontrolovaná verzia klientskej aplikácie a jej status bajt. Obraz firmvéru klientskej aplikácie má pevne stanovenú adresu v pamäti 0xB000, respektíve nachádza sa na začiatku jedenásteho bloku v pamäti. Hlavička OADHeader sa získa funkciou *memcpy()*.

```
uint8_t headerBuff[16];
memcpy(headerBuff, (uint32_t *)((11 * 4096)), 16);
```

OADClient sa bude pokúšať pripojiť k OADServeru pomocou pravidelne vysielaných jednoduchých správ. Je pravdepodobné, že v reálnych podmienkach bude server odpovedať okamžite, je však možné, že práve bude stahovať novú aktualizáciu, preto sú správy vysielané v sekundových intervaloch. Časovač sa nastavuje pomocou SimpleLink SDK, vďaka čomu je zabezpečená jednoduchá modifikácia kódu pre iné mikrokontroléry z rodiny CC Texas Instruments.

```
/* Vytvorenie objektu so základnými parametrami časovača */
Clock_Params clkParams;
Clock_Params_init(&clkParams);

/* Nastavenie parametrov časovača */
clkParams.period = REPORTINTERVAL_DURIATION_MS * 1000 /
    Clock_tickPeriod;
clkParams.startFlag = FALSE;
Clock_construct(&reportTimeout, timerCallback, 1, &clkParams);
reportTimeoutHandle = Clock_handle(&reportTimeout);

...

/* Spustenie časovača */
Clock_start(reportTimeoutHandle);
```

Ukážka 6.1: Nastavenie časovača

Na riadku 8 ukážky 6.1 sa nastavuje funkcia, ktorá je pravidelne volaná pri vypršaní časovača. Funkcia *timerCallback()* zavolá udalosť v hlavnej slučke programu, ktorý odošle správu. Po prijatí potvrdzovacej správy server odošle hlavičku OADHeader s informáciou o dostupnej aktualizácii. Ak je dostupný nový obraz firmvéru, začne sa proces sťahovania.

```
#define OADTarget_BUILD_UINT16(loByte, hiByte) \ ((uint16_t)(((
    loByte) & 0x00FF) + (((hiByte) & 0x00FF) << 8)))

/* Získanie celkovébo počtu blokov na stiahnutie */
oadBlkTot = OADTarget_BUILD_UINT16(pValue[hdrOffset + 2], pValue[
    hdrOffset + 3]) / (OAD_BLOCK_SIZE / HAL_FLASH_WORD_SIZE);
```

Ukážka 6.2: Získanie veľkosti updatu

Prijímaný obraz nesmie byť vačší ako je veľkosť dostupnej pamäte, v opačnom prípade sa aktualizácia neuskutoční. Rovnako je to aj prípade, že verzia obrazu je zhodná s aktuálnym obrazom. Pokiaľ sú tieto podmienky splnené,

klient inicializuje sťahovanie požiadavkou na prvý blok obrazu. Pred odoslaním je potrebné zostaviť payload. Pri požiadavke o blok payload obsahuje iba číslo požadovaného bloku.

```
uint8_t *msgBuffer;
/* Pred vytvorením payloadu je potrebné alokovať priestor */
msgBuffer = (uint8_t*) malloc(msgLen);
if(msgBuffer == NULL)
{
    return NULL;
}
memset(msgBuffer, 0, msgLen);
```

Ukážka 6.3: Alokovanie priestoru pre payload

Po alokovaní pamäte, ako je zobrazené v ukážke 6.3, sa prvému bajtu priradí typ správy, v tomto prípade *PACKET_TYPE_OAD_BLOCK_REQ*. Ten predstavuje hodnotu 0x06 podľa tabuľky 5.3.

```
/* Typ správy informuje o požiadavke čísla bloku */
pOadBlockReqPacket[0] = PACKET_TYPE_OAD_BLOCK_REQ;

/* Číslo bloku má 2 bajty*/
pOadBlockReqPacket[2] = blockNum & 0xFF;
pOadBlockReqPacket[3] = (blockNum >> 8) & 0xFF;
```

Ukážka 6.4: Vytvorenie payloadu správy

```
/* Nastavenie adresy prijímateľa */
currentRadioOperation.easyLinkTxPacket.dstAddr[0] = dstAddr;
/* Nastavenie adresy odosielateľa */
pPacket[RADIO_PACKET_SRCADDR_OFFSET] = address;
/* Vloženie payloadu */
memcpy(currentRadioOperation.easyLinkTxPacket.payload, pPacket,
    packetLen);
/* Veľkosť správy */
currentRadioOperation.easyLinkTxPacket.len = packetLen;

/* Odoslanie správy  */
if (EasyLink_transmit(&currentRadioOperation.easyLinkTxPacket) !=
     EasyLink_Status_Success) {
    System_abort("EasyLink_transmit failed");
}
```

Ukážka 6.5: Odoslanie správy

Po odoslaní požiadavky klient očakáva odpoveď od servera typu *PACKET_TYPE_OAD_BLOCK_RSP*, ktorá obsahuje 64 bajtov bloku nového

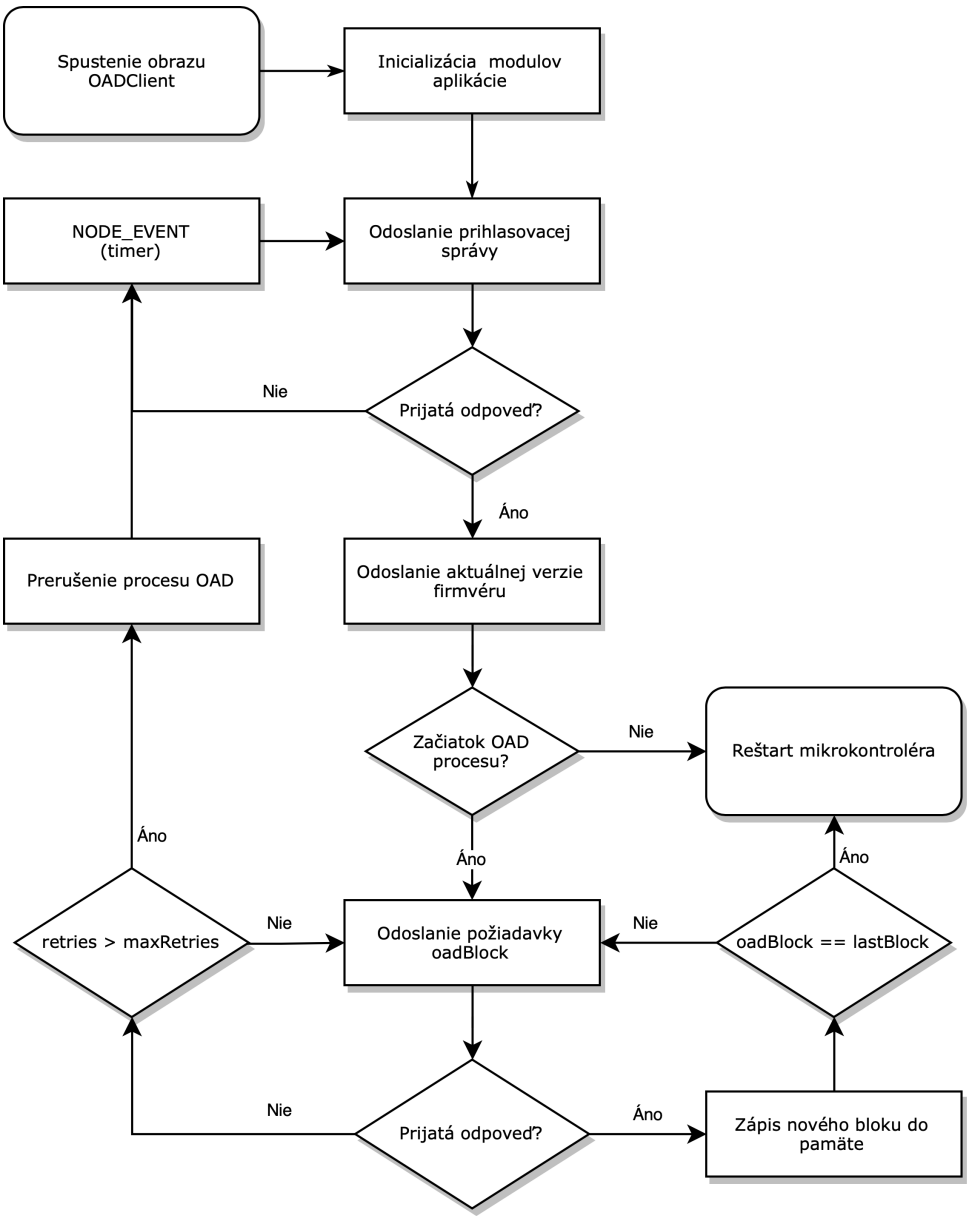
obrazu. Zápis do pamäte sa uskutočňuje pomocou funkcie *FlashProgram()*, pričom jej vstupmi sú ukazovateľ na miesto v pamäti, kde sú prijaté dáta, adresa zápisu a počet slov. Zápis na každú adresu pozostáva z výsledku AND bitovej operácie novej hodnoty a existujúcich dát. Dáta môžu byť prepisované neobmedzene, avšak z uvedeného vyplýva, že nie je možné zmeniť logickú 0 v pamäti na logickú 1. Pred zápisom je preto potrebné blok pamäte vymazať pomocou *FlashSectorErase()*. Taktiež je výrobcom odporúčané pred zápisom deaktivovať vyrovnávaciu pamäť, keďže nie je automaticky aktualizovaná v prípade nezhody s novým obsahom Flash pamäte. Inštrukcie prepisovania pamäte Flash sa nachádzajú v ROM a je nežiadúce, aby sa v momente zápisu vykonávali iné inštrukcie z pamäte Flash, čo môže mať za následok fatálne zlyhanie celého programu. Preto je nutné pred zápisom deaktivovať všetky prerušenia.

```
1  while (len) {
2
3    if (len > 8) {
4      writeIncrement = 8;
5    } else {
6      writeIncrement = len;
7    }
8
9    /* Deaktivácia prerušení */
10   key = Hwi_disable();
11   /* Zápis do pamäte */
12   status = FlashProgram(pBuf, (uint32_t)FLASH_ADDRESS(page,
       offset), writeIncrement);
13   Hwi_restore(key);
14
15   if (status != 0) {
16     while(1); //Error writing to flash
17   } else {
18     len -= writeIncrement;
19     pBuf += writeIncrement;
20     offset += writeIncrement;
21   }
22 }
```

Ukážka 6.6: Zápis do pamäte Flash

Na ukážke 6.6 je znázornená rutina zapisovania do pamäte. Premenná *len* predstavuje celkovú veľkosť zapisovaných dát. Počas zápisu sú zastavená prerušenia, preto je dobré pre zlepšenie vlastností zápis rozdeliť na viacero menších blokov. Na konci každej iterácie sa ukazateľ na prijaté dáta *pBuf* inkrementuje rovnako ako *offset*, ktorý ukazuje na adresu v pamäti.

Po prijatí a zapísaní posledného bloku sa končí úloha aplikácie OADClient. Resetovanie MCU je pomerne jednoduché a zabezpečuje ho funkcia *SysCtrlSystemReset()*. *SysCtrlSystemReset()* ukončí všetky spustené funkcie, prerušenia a spôsobí celkový reštart zariadenia. Po reštarte sa už spustí bootloader s novým obrazom firmvéru.



Obrázok 6.1: Vývojový diagram aplikácie OADClient

6.2 OADServer

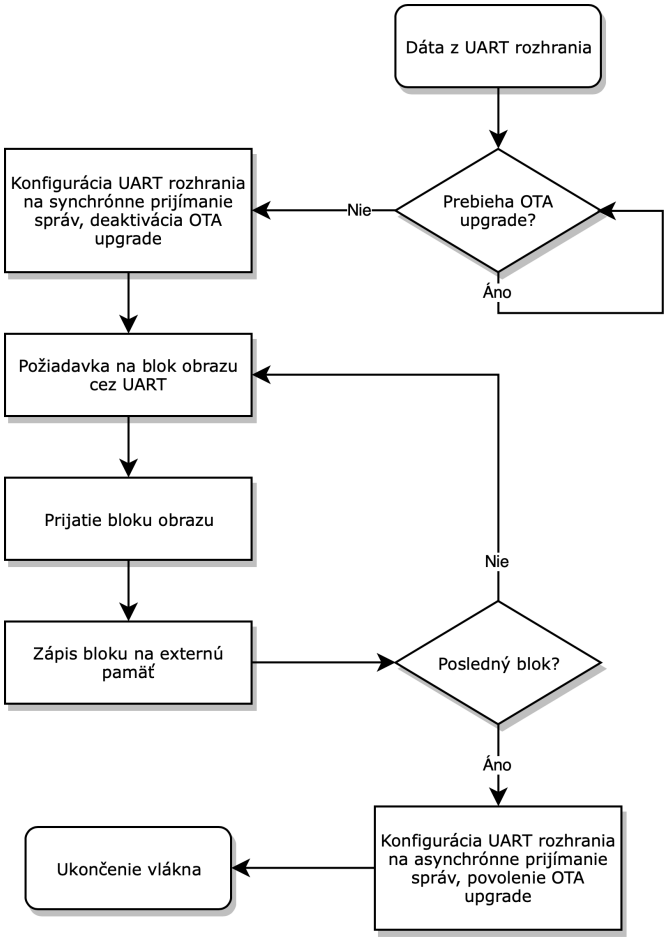
Serverová časť aplikácie poskytuje nový firmvér klientom. Serverové zariadenie (hub) pozostáva z LaunchPad-u CC1310 a počítača Raspberry Pi. LaunchPad CC1310 obsahuje 8 Mb pamäť, ktorá je využitá na uloženie nového obrazu firmvéru. Hub prijme informáciu z externého servera o dostupnej aktualizácii a po stiahnutí je tento obraz pomocou UARTu nahraný na externú Flash pamäť. V ukážke 6.7 je popísané nastavenie UART komunikácie mikropočítača Raspberry Pi s mikrokontrolérom CC1310. Hodnota *UART_MODE_CALLBACK* parametra readMode zabezpečí asynchrónne zavolanie funkcie v parametri readCallback. Priebeh prijímania nového obrazu je naznačený na obrázku 6.2.

```
1  UART_Params uartParams;
2  /* Inicializácia UART */
3  UART_Params_init(&uartParams);
4  uartParams.writeDataMode = UART_DATA_BINARY;
5  uartParams.readDataMode = UART_DATA_BINARY;
6  uartParams.readReturnMode = UART_RETURN_FULL;
7  uartParams.readEcho = UART_ECHO_OFF;
8
9  /* Nastavenie rýchlosti prenosu */
10 uartParams.baudRate = 115200;
11
12 /* Nastavenie asynchrónnej komunikácie */
13 uartParams.readMode = UART_MODE_CALLBACK;
14 uartParams.readCallback = OADServer_callbackImgHeaderFromUART;
15
16 /* Inicializácia UART rozhrania */
17 uartHandle = UART_open(Board_UART0, &uartParams);
18
19 /* Neblokujúca funkcia */
20 UART_read(uartHandle, pImgInfo, 16);
```

Ukážka 6.7: Nastavenie UART

Samotný prenos obrazu prebieha na podobnej báze ako bezdrôtový prenos medzi klientom a serverom. Obraz firmvéru je posielaný po 64 bajtových blokoch. Každý blok si aplikácia vyžiada po tom, čo do pamäte zapíše predošlý. Na zápis to externej pamäte flash je použitá funkcia *ExtFlash_write()* knižnice *ExtFlash.h* a na zápis využíva Serial Peripheral Interface (SPI).

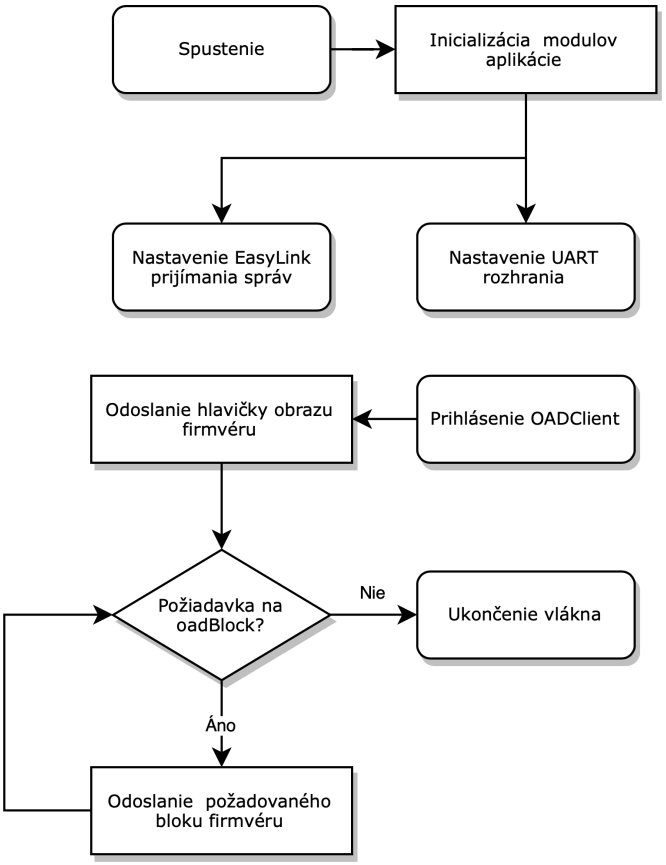
Po uložení obrazu na externú flash pamäť je server pripravený k bezdrôtovej aktualizácii klientov alebo prijatiu nového obrazu. Celkový priebeh aplikácie OADServer je zobrazený na diagrame 6.3. Update môže začať až keď sa na



Obrázok 6.2: Vývojový diagram UART komunikácie

zariadení spustí OADClient. K tomu je potrebné, aby firmvér bol schopný prijať informáciu o dostupnosti novej aktualizácie, nastavil správnu hodnotu parametra state v hlavičke firmvéru a inicializoval reštart zariadenia do bootloaderu.

Ak server prijme prihlasovaciu informáciu od klienta, odošle hlavičku firmvéru aktuálne dostupného obrazu. V prípade, že sa na zariadení nachádza starší firmvér, inicializuje prenos požiadavkou na prvý blok obrazu. Počas aktualizácie jedného zariadenia sú prípadné ďalšie zariadenia ignorované. V rámci architektúry je projekt pripravený vykonávať paralelnú aktualizáciu viacerých koncových zariadení naraz, avšak vrámci súčasnej štruktúry projektu táto funkcia nie je žiadúca.



Obrázok 6.3: Vývojový diagram aplikácie OADServer

Kapitola 7

Záver

Hlavnou témou tejto diplomovej práce bola bezdrôtová aktualizácia embedded zariadení. Po analýze v Kapitole 3 aktuálne používaných metód je možné skonštatovať, že bezdrôtový proces sa používa výrazne viac spolu s rozšírením platformy IoT, nakoľko je oproti klasickej aktualizácii s využitím fyzického pripojenia pohodlnejší a praktickejší.

Pri bezdrôtovej aktualizácii firmvéru sa môžu využívať rôzne technológie, častou voľbou je Bluetooth, ktorý je bežne podporovaný a od verzie 4.2 podporuje šifrovacie mechanizmy pre zvýšenie bezpečnosti aktualizácie. Práve bezpešnosť je totižto nesmierne dôležitým aspektom procesu aktualizácie, keďže môže dôjsť k úniku údajov alebo poškodeniu koncového zariadenia. Proti týmto hrozbám sa dá brániť rôznymi spôsobmi, ktoré boli naznačené v Sekcii 3.1.

Nasledujúcim krokom bol samotný návrh pre konkrétny mikrokontrolér TI CC1310. Splnenie požiadavky na nízku spotrebu je zaručené najmä výberom bezdrôtovej technológie Sub-1GHz, ktorá sa ňou vyznačuje. Ďalším spôsobom, akým by bolo možné znížiť spotrebu, bolo využiť inkrementálny upgrade. Tento postup by bol ale pre túto aplikáciu veľmi problematický kvôli rozdeleniu pamäte na 4 KB bloky. Pri rozdielnej dĺžke kódu by tak dochádzalo k posunu medzi blokmi.

Výsledný návrh bol implementovaný a odskúšaný v reálnych podmienkach. Pri opakovanom testovaní sa aplikácia OTA ukázala ako stabilná a spoľahlivá,nedochádzalo teda k žiadnym chybám pri prenose obrazu, ani pri jeho následnom zavedení do prevádzky.

Riešenie je tak značným pokrokom vpred oproti predošlej situácii s aktu-

alizáciou používaných modulov. Tie sa doteraz vzhľadom na zlú prístupnosť v mieste použitia a veľké množstvo aktualizovali iba v prípade chýb ohrozujúcich základnú funkčnosť alebo bezpečnosť. Zároveň je OTA aktualizácia implementovaná vrámci doteraz používaného komunikačného protokolu a proprietárneho šifrovania.

Vzhľadom na použitie technológie Sub-1GHz bola tiež dosiahnutá nízka spotreba a veľký dosah. Počas vývoja tohto riešenia bolo paralelne iným tímom navrhnutý OTA upgrade s využitím technológie Bluetooth. Táto možnosť je ale nedostatočná oproti diskutovanej implementácii kvôli nízkemu dosahu technológie. S využitím Sub-1GHz sa môže na aktualizáciu využiť hub slúžiaci priamo na získavanie dát od modulov.

Tento návrh môže tiež slúžiť ako základný stavebný blok pre ďalšie rozšírenia OTA aktualizácie. Síce to aktuálne nie je potrebné, ale navrhnutá implementácia môže byť v budúcnosti jednoducho rozšírená pre paralelný upgrade viacerých koncových zariadení. Ďalej sa môže implementovať možnosť aktualizácie aplikácie OADClient, poprípade ušetriť miesto na pamäti odstránením TI-RTOS v aplikácii OADClient a využiť čisto API EasyLink.

Okrem predošlých spomenutých vlastností je proces OTA aktualizácie firmvéru ošetrený tak, aby nedošlo ku znefunkčneniu modulov. Pokiaľ je nesprávne prijatý akýkoľvek paket od servera, zahodí sa a klient vyšle požiadavku o ten istý paket. V prípade neúspešného prijatia alebo zavedenia firmvéru si klient neustále žiada o nový obraz firmvéru. Ten je tiež okrem proprietárneho šifrovania na záver chránený kontrolným súčtom CRC16.

S ohľadom na všetky uvedené argumenty je možné tvrdiť, že konečný návrh OTA aktualizácie MCU TI CC1310 splňuje potrebné vlastnosti na použitie v praxi.

Prílohy

Dodatok A

Zoznam skratiek

AON Always-On.

AP Access point. Prístupový bod.

API Application Programming Interface. Rozhranie pre programovanie aplikácií.

BLE Bluetooth Low Energy.

CRC Cyclic redundancy check. Cyklická kontrola.

IDE Integrated Development Environment. Integrované Vývojové Prostredie

IEEE Institute of Electrical and Electronics Engineers. Inštitút pre elektrotechnické a elektronické inžinierstvo.

IoT Internet of Things. Internet vecí.

ISM Industrial Scientific and Medical radio bands. priemyslové vedecké a zdravotnícke radiové pásma.

JTAG Joint Test Action Group. Permanentná pamäť.

MCU Microcontroller Unit.

OAD Over the Air Download.

OTA Over The Air.

RFID Radio Frequency Identification. Vysokofrekvenčná identifikácia.

RISC Reduced instruction set computer. Procesory s redukovanou inštrukčnou sadou.

ROM Read-Only Memory. Permanentná pamäť.

RTC Real-time clock. Hodiny reálného času.

RTOS Real Time Operating System. Operačný systém reálneho času

SDK Software Development Kit. Sada vývojových nástrojov

SPI Serial Peripheral Interface. Sériové periférne rozhranie

TLS Transport Layer Security.

WEP Wired Equivalent Privacy. Súkromie ekvivalentné drôtovým sieťam.

WPA Wi-Fi Protected Access. Chránený prístup k wifi.

WSN Wireless Sensor Networks. Bezdrôtové senzorové siete.

Dodatok B

Literatúra

[1] Shayaan Jagtap. IoT + Machine Learning is Going to Change the World, 2018. Dostupné z https://towardsdatascience.com/iot-machine-learning-is-going-to-change-the-world-7c4e0cd7ac32.

[2] Padraig Scully. The Top 10 IoT Segments in 2018, 2001. Dostupné z https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/.

[3] Digi Key Electronics. Texas instruments LAUNCHXL-CC1350US, 2019. Dostupné z https://www.digikey.ca/product-detail/en/texas-instruments/LAUNCHXL-CC1350US/296-44892-ND6217829.

[4] Texas Instruments Incorporated. Simplelink CC13x0 Proprietary RF User's Guide, 2019. Dostupné z http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_60_00_21/docs/.

[5] Dave Evans. The Internet of Things, 2011. Dostupné z https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[6] Kyoochun Lee In Lee. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *ScienceDirect*, 10, December 2015.

[7] RFID Inc. Radio Frequency Identification, 2019. Dostupné z https://www.rfidinc.com/rfid-101/.

[8] IEEE. GET 802(R) Standards, 2019. Dostupné z https://ieeexplore.ieee.org/browse/standards/get-program/page/series?id=68.

[9] National Institute of Standards and Technology. ADVANCED ENCRYPTION STANDARD, 2001. Dostupné z https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[10] Bluetooth Inc. Our History, 2019. Dostupné z https://www.bluetooth.com/about-us/our-history/.

[11] plugintoiot. What is Sub 1Ghz RF, its range & the importance of Sub 1Ghz for wireless IoT applications, 2016. Dostupné z https://www.plugintoiot.com/sub-1-ghz/.

[12] Jessica Califano. 5 Reasons to Use Sub-GHz for IoT Applications, 2018. Dostupné z https://blog.temboo.com/using-sub-ghz-for-iot-applications/.

[13] Ondrej Kachman. Effective Multiplatform Firmware Update Process for Embedded Low-Power Devices, 2018. Dostupné z http://acmbulletin.fiit.stuba.sk/abstracts/kachman2018.pdf.

[14] Jan Šimůnek. Secure Firmware Upgrade of Embedded Platform, 2018. Dostupné z https://dspace.cvut.cz/bitstream/handle/10467/76736/F3-BP-2018-Simunek-Jan-Secure_Firmware_Upgrade_of_Embedded_Platform.pdf.

[15] Benjamin Bucklin Brown. Over-the-Air (OTA) Updates in Embedded Microcontroller Applications, 2018. Dostupné z https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html.

[16] IEEE. IEEE Standard for Test Access Port and Boundary-Scan Architecture, 2017. Dostupné z https://ieeexplore.ieee.org/document/6515989.

[17] Richa Dham. Over-the-air firmware upgrades for internet of things devices, 2017. Dostupné z https://www.embedded-computing.com/embedded-computing-design/over-the-air-firmware-upgrades-for-internet-of-things-devices.

[18] Bc. Tomáš Minár. *Návrh systému pro vzdálený upgrade firmwaru pro uzly bezdrátové senzorové sítě*. Vysoké učení technické v Brně, Fakulta elektro-techniky a komunikačních technologií, Ústav telekomunikací, 2014.

[19] STMicroelectronics. The BlueNRG-1, BlueNRG-2 BLE (OTA) (over-the-air) firmware upgrade, 2018. Dostupné z

https://www.st.com/content/ccc/resource/technical/document/application_note/group0/09/72/f9/6b/d7/8f/48/fd/DM00293821/files/DM00293821.pdf/jcr:content/translations/en.DM00293821.pdf.

[20] Cypress Semiconductor Corporation. Over-the-Air (OTA) Firmware Update Procedure for Bluetooth Low Energy (BLE) Devices, 2017. Dostupné z https://www.cypress.com/file/298486/download.

[21] Texas Instruments Incorporated. CC1310 simplelink Ultra-Low-Power Sub-1 GHz Wireless MCU, 2019. Dostupné z http://www.ti.com/lit/ds/symlink/cc1310.pdf.

[22] CSc. Doc. Ing. Zdenék Kotásek. Architektury CISC a RISC, uplatnění v personálních počítačích, 2019. Dostupné z https://www.fit.vutbr.cz/study/courses/ITP/public/itp07/architektury01.pdf.

[23] Arm Limited. T32 Instruction Set, 2019. Dostupné z https://developer.arm.com/architectures/instruction-sets/base-isas/t32.

[24] Prabal Dutta. ARM Architecture Overview, 2019. Dostupné z https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/ARM_Architecture_Overview.pdf.

[25] National Instruments. What is a Real-Time Operating System (RTOS)?, 2019. Dostupné z http://www.ni.com/cs-cz/innovations/white-papers/07/what-is-a-real-time-operating-system–rtos–.html.

[26] WA&S Ltd. What is an RTOS?, 2019. Dostupné z https://www.highintegritysystems.com/rtos/what-is-an-rtos/.

[27] Amazon Web Services, Inc. What is an RTOS?, 2019. Dostupné z https://www.freertos.org/about-RTOS.html.

[28] Texas Instruments Incorporated. TI-RTOS: Real-Time Operating System (RTOS) for Microcontrollers (MCU), 2019. Dostupné z http://www.ti.com/tool/TI-RTOS-MCU.

[29] Texas Instruments Incorporated. Code Composer Studio (CCS) Integrated Development Environment (IDE), 2019. Dostupné z http://www.ti.com/tool/CCSTUDIO.

[30] Texas Instruments Incorporated. Simplelink-Easylink, 2019. Dostupné z http://processors.wiki.ti.com/index.php/SimpleLink-EasyLink.