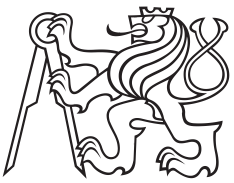
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra telekomunikační techniky

Bezdrátový komunikační systém pro turistické průvodce

Karel Zadražil

Květen 2018
Vedoucí práce: Ing. Pavel Troller, CSc.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zadražil** Jméno: **Karel** Osobní číslo: **392872**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

Studijní obor: **Komunikační systémy a sítě**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Bezdrátový komunikační systém pro turistické průvodce

Název diplomové práce anglicky:

Wireless Communication System for Tourist Guides

Pokyny pro vypracování:

Navrhněte a zkonstruujte prototyp zařízení pro turistické průvodce, umožňující následující funkce:
- Jednosměrný přenos výkladu průvodce do sluchátek turistů - klientů až do celkového počtu 50 klientů. Dosah komunikace cca 50 m na přímou viditelnost s občasnými překážkami
- Zpětná informace průvodci při ztracení klienta z dosahu či při stisknutí tlačítka požadavku klientem
- Doba provozu všech komponent zařízení minimálně 6 hodin na jedno nabití
- Možnost připojení mikrofonu (živý výklad) a reprodukce předem nahraného zvukového souboru (který uslyší i průvodce pro svoji synchronizaci s nahrávkou).

Výhodné bude využití chytrého telefonu jako vysílací stanice a možnost příjmu rovněž pomocí speciální aplikace na klientských telefonech, nicméně konstrukce speciálního malého jednoúčelového přijímače je žádoucí.

Seznam doporučené literatury:

[1] Bensky, A.: Short-range Wireless Communication, Second Edition: Fundamentals of RF System Design and Application (Communications Engineering Series). Elsevier 2004. ISBN 0-750-67782-1.
[2] Kraemer, R.; Katz M.D.: Short-Range Wireless Communications: Emerging Technologies and Applications. Wiley Online Library 2009. ISBN: 978-0-47074-012-5. Dostupné na http://onlinelibrary.wiley.com/book/10.1002/9780470740125 [on-line]

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Pavel Troller, CSc., katedra telekomunikační techniky FEL

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

Datum zadání diplomové práce: **05.01.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

[redacted]
Ing. Pavel Troller, CSc.
podpis vedoucí(ho) práce

[redacted]
podpis vedoucí(ho) ústavu/katedry

[redacted]
prof. Ing. Pavel Ripka, CSc.
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.

23.2.2018
Datum převzetí zadání

[redacted]
Podpis studenta

Poděkování / Prohlášení

Na tomto místě bych rád poděkoval vedoucímu práce za cenné rady a také své rodině za klidné pracovní podmínky.

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

Abstrakt / Abstract

Tato práce se zabývá problematikou zpracování a přenosu zvuku pomocí IP sítě a následné reprodukce na druhém vzdáleném zařízení. Zachytávání zvuku probíhá na mobilním telefonu a vlastní reprodukce na separátním HW zařízení.

Klíčová slova: ESP8266, UDP, přenos zvuku, aplikace pro chytré telefony, I2S, PWM, broadcast, WiFi moduly

This thesis deals with the processing and transmission of sound using the IP network and reproduction on the second remote device. The cell phone captures the sound and playback is performed on a dedicated HW device.

Keywords: ESP8266, UDP, audio transmission, smartphone applications, I2S, PWM, broadcast Wifi modules

Title translation: Wireless Communication System for Tourist Guides

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Zadání této diplomové práce vzniklo na základě reálných potřeb průvodců. Ti mnohdy potřebují hovořit k velké skupině lidí, což se stává (bez použití dalších technických prostředků) s rostoucím počtem turistů stále namáhavější. Užití megafonu mnohdy není možné z více důvodů. Tím pádem se sety vysílaček se sluchátky jeví jako ideální volba i do míst, kde je potřeba nerušit ostatní hlasitým výkladem.

Podobná zařízení na trhu samozřejmě existují, ovšem buďto pracují na analogovém principu a nejsou tím pádem použitelná v zarušeném prostřední, nebo se jedná o velmi drahé přístroje.

Cílem práce je tedy vytvořit takovou sadu, jejíž koncová zařízení budou:

- na pohled nezajímavá (kvůli odcizení)
- uživatelsky přívětivá
- odolná proti rušení
- levná
- lehká
- malá

Na průvodcovskou stanici již nejsou kladeny tak přísné nároky. Přesto jsem se snažil pokud možno maximálně vyhovět zadání, proto jsem se vydal cestou mobilní aplikace.

Kapitola 2
Výběr HW

2.1 Mikroprocesor

2.1.1 Obecné vlastnosti

Pro daný účel jsem vybral procesor ESP8266 firmy Espressif s integrovaným Wi-Fi transceiverem. Čip byl původně navržen jako WiFi – UART převodník. Tomu ostatně odpovídá i firmware, se kterým je většinou dodáván – ovládá se AT příkazy přes vestavěný UART port. Velké množství elektrotechniků však v ESP8266 spatřilo mnohem větší potenciál, a tak kolem ESP8266 vznikla komunita, která pro něj našla nejrůznější využití.

Mezi hlavní přednosti tohoto čipu patří především:

- velice nízká cena
- dostupnost v modulech
- malé rozměry (čipu i modulů)
- nízká spotřeba
- vysoký výpočetní výkon
- mnoho programovacích jazyků
- snadné přeprogramování
- RTC časovač funguje i v režimu hlubokého spánku

Hlavní nevýhody jsou:

- špatná dokumentace
- malá paměť
- nízký počet I/O pinů

2.1.2 Moduly

Nyní si jednotlivé položky rozebereme detailněji. Tento čip se dá pořídit již za 2 americké dolary včetně poštovného (v době tvorby této práce dostupný na `www.ebay.com` či `www.aliexpress.com`). Za tuto cenu ovšem nedostanete pouze samotný čip, nýbrž celý modul. Není potřeba se tedy trápit návrhem či výběrem antény a jejím impedančním přizpůsobením. Moduly jsou osazeny SPI flash pamětí. Typicky to bývá 512 KB – 4 MB. To je velmi důležité, jelikož sám procesor žádnou permanentní přepisovatelnou pamětí nedisponuje. Když už jsme u paměti, šetřilo se i RAM. Pokud je připojen k AP, pak pro vlastní program zbývá kolem 40 KB. Nechybí ani krystal (většinou 40 MHz) a kondenzátory na napájení. Mnohdy na destičce najdeme i jednu či dvě LED diody. Jedna bývá připojená na GPIO2 a pokud má modul dvě, druhá indikuje přítomnost napájení (např. ESP-01). Modul je tedy po připojení napájení schopný samostatné činnosti.

Na trhu existuje celá řada modulů (řádově desítky) a stále přibývají nové. Většina z nich má meandrovou anténu přímo na PCB, což je nejlevnější varianta. Nejmenší moduly žádnou anténu nemají (například ESP-04 který na PCB zabere pouhých 14,7 x 12.1 mm). Jiné disponují U.FL konektorem (např. modul ESP-02) nebo dokonce U.FL konektorem v kombinaci s keramickou anténou (např. ESP-07).

2.1.3 Výkon a spotřeba

Průměrná spotřeba modulu se v módu klient pohybuje okolo 35 mA. V módu AP je to už kolem 80 mA. Je však důležité mít na zřeteli, že se jedná o průměrnou spotřebu. Špičkový proud během vysílání dosahuje hodnot kolem 300 mA. Úměrně tomu je potřeba dimenzovat napájení.

Na křemíku se ukrývá 32-bitové výpočetní jádro taktované na 80 MHz. Přestože musíme počítat s tím, že obsluha Wi-Fi si odkrojí jistou část výkonu, pořád nám zbývá dostatek prostředků pro běh vlastního programu.

2.1.4 Firmware

Pokud se rozhodneme nezůstat u originálního firmwaru založeném na AT příkazech, potom máme výběr z opravdu širokého množství jazyků. Je možné do ESP modulu nahrát interpret jazyka Lua. To se skvěle hodí pro rychlý vývoj a ladění, nebo když jednoduše potřebujeme nějaký senzor či aktuátor velice rychle připojit do sítě. Existuje dokonce interpret jazyka MicroPython který má velmi obdobné využití. Také byl vyvinut doplněk do Arduino IDE, takže vývojové desky osazené převodníkem je možné programovat téměř tak jednoduše, jako kdyby se jednalo o klasické Arduino. Samozřejmě nechybí ani vývojové prostředí pro jazyk C, který vyniká především výkonností přeložených programů. SDK pro jazyk C vyvinula přímo firma Espressif Systems. Ačkoliv se jedná o vyšší programovací jazyk, pořád má poměrně blízko k HW.

Přeprogramování modulu novým firmwarem je velice jednoduché. Stačí při zapnutí čipu (případně během resetu) oproti běžnému zapojení stáhnout GPIO0 do logické nuly (0 V). Tím se nezačne vykonávat program z připojené SPI flash paměti, ale místo toho přijde na řadu bootloader pevně uložený v ROM paměti přímo v čipu. Ten očekává nahrání nového firmwaru po sériové lince, který ukládá právě do připojené SPI flash paměti.

Další možností je využití OTA. Základní princip je poměrně jednoduchý. Program je uložen pouze na jedné polovině SPI paměti. V něm je také obsažen mechanismus získání nového firmware. Je už na nás, jestli tím mechanismem bude aktivní dotazování se nějakého HTTP serveru či naslouchání na určitém portu. Nový firmware se postupně ukládá do zatím prázdné části flash paměti. Pokud je přenos úspěšný, zařízení přepne využívání paměti na onu nově zapsanou a dosud nevyužívanou polovinu a poté se resetuje, čímž začne vykonávat nový program. OTA s sebou přináší velké zjednodušení, kdy vůbec nemusíme mít fyzický přístup k danému zařízení a přesto jsme schopni velmi jednoduše nahrát nový firmware. To se hodí především tam, kde by vyvedení programovacích pinů ven z krabičky nebylo vhodné, tam kde není k dispozici potřebné vybavení (napři USB – UART převodník), kde je zařízení obtížně fyzicky nedosažitelné nebo takových zařízení potřebujeme aktualizovat desítky či stovky. Mezi hlavní nevýhody OTA patří pouze poloviční reálná využitelnost flash paměti pro program a také nutnost dobrého zabezpečení celého OTA.

2.1.5 Dokumentace

Největším negativem tohoto čipu je velmi špatná dokumentace. Celý katalogový list má 28 stran [1]. Jen pro porovnání, obdobné mikrokontroléry mají běžně katalogové listy o stovkách stran. Pokud se tedy budete pídit po nějaké velice konkrétní informaci, s největší pravděpodobností ji v oficiální dokumentaci nenajdete. Naštěstí existuje spousta nadšenců, kteří si ESP moduly oblíbili právě pro svou nízkou pořizovací cenu a metody reverzního inženýrství jim nejsou cizí. Většinu důležitých informací tedy lze s větší či menší věrohodností získat na různých internetových fórech [2–4].

Samostatnou kapitolou je dokumentace modulů. U nich si musíte dát pozor i na tak základní věc, jako je velikost instalované paměti. Některé moduly se totiž vyrábějí ve více variantách. Ačkoliv se to může zdát až úsměvné, pak špatným nápadem není překontrolování popisků I/O pinů. Vyhnete se tak rozčarování v podobně nefungujícího zapojení, nebo v horším případě spálenému obvodu. U samostatných modulů jsem se zatím s tímto nešvarem nesetkal, ale například u vývojové desky NodeMCU je potřeba si na to dát pozor.

2.1.6 GPIO a funkce

ESP8266 má dohromady 17 GPIO pinů. Jak již ale bylo zmíněno, čip samotný potřebuje externí paměť pro program. Tím je trvale obsazeno 6 I/O pinů. Ani zbylých 11 GPIO nelze použít zcela libovolně.

GPIO0 slouží při startu k přepnutí do programovacího režimu. Naopak GPIO15 je potřeba při startu přidržet v logické nule. V opačném případě se bude ESP8266 pokoušet zavést program z SD-karty. Pinem označovaným jako CH_PD se, jak už jeho název napovídá, obvod deaktivuje. Pro aktivní stav je nutné jeho připojení na logickou jedničku. Slovní popis může být poněkud zmatečný. Pokud se podíváte do katalogového listu [1], o moc moudřejší také nebudete. Vše nejlépe objasní tabulka 2.1.

Mód startu	GPIO0	GPIO2	GPIO15
Zavedení z FLASH paměti	1	1	0
Programovací režim (přes UART)	0	1	0
Zavedení z SD karty	X	X	1

Tabulka 2.1. Tabulka funkcí GPIO pinů ESP8266. Převzata z [5] a opravena podle skutečného chování čipu.

Kromě již zmíněných funkcí na GPIO pinech najdeme: 1x analogový vstup, ještě jeden UART a I2S sběrnici.

Čip disponuje třemi spánkovými režimy. V nejhlubším běží samotné RTC hodiny, které ovšem neumí nic jiného, než své vypršení oznámit na GPIO16 logickou nulou. Toho se dá ovšem s výhodou využít propojením GPIO16 s resetem. Tímto opatřením se celý modul po uplynutí časovače resetuje a taktéž probudí.

2.2 Baterie

Jako dostupný a levný zdroj energie jsem zvolil sekundární li-ion články. Mají vysokou hustotu energie a také relativně vysoké nominální napětí (cca 3,7 V), což je pro napájení zařízení pracujícího v rozmezí 2,5 – 3,6 V téměř ideální.

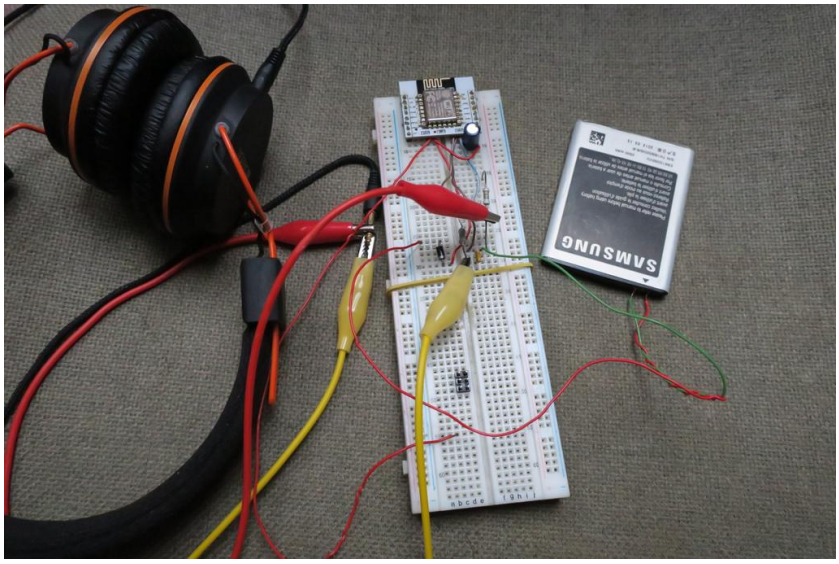
2.3 Regulátor napětí

Vybral jsem opět velmi levný lineární stabilizátor HT7333 [6], který splňuje nároky na nízký minimální úbytek napětí a také dokáže dodat dostatečný proud. Malý úbytek je důležitý především kvůli napájení z li-ion článků, kdy zcela vybitý článek má kolem 2,5 V. To je zároveň nejnižší napětí, při kterém je ESP schopné pracovat. Čím vyšší tedy bude minimální úbytek na regulátoru, tím bude větší procento energie, které v baterii zůstane při plném vybití.

Většina ESP modulů má vývody s roztečí 2 mm, což v důsledku znamená, že pokud budete chtít zapojení testovat na nepájivém kontaktním poli, budete potřebovat adaptér. Milou výhodou HT7333 je fakt, že rozmístění pinů v pouzdře SOT-89 přesně odpovídá ploškám pro volitelné osazení na adaptéru regulátorem. Stačilo tedy zapájet regulátor a odstranit zkratovací propojku.

2.4 Prvotní testování

Před započetím další práce jsem se rozhodl alespoň rámcově otestovat HW část, abych si byl jistý tím, že je můj záměr proveditelný. Jak je patrno z předchozí části této kapitoly, ESP8266 nedisponuje žádným analogovým zvukovým výstupem. Použití DAC by celé zapojení prodražilo a zkomplikovalo, budeme se mu tedy snažit vyhnout. Cílem je prověřit, zda kombinace stabilizátoru, baterie z telefonu a zvukového PWM výstupu bude kompatibilní a poskytne rozumnou kvalitu poslechu.



Obrázek 2.1. Pohled na zapojení při prvních testech na nepájivém poli

2.4.1 Software

Nejprve bylo potřeba nainstalovat SDK [7] firmy Espressif Systems, které umožňuje kompilovat programy napsané pro ESP8266 v jazyce C. Zde jsem postupoval podle podrobného návodu na Githubu [8].

Jako testovací program jsem zvolil MP3 dekodér pro ESP8266 [9], jelikož vyžaduje jen drobné úpravy k plnému zprovoznění. Pro maximální kvalitu program počítá s připojením externího DAC a SPI RAM. Drobnou úpravou kódu lze použití externí RAM paměti vypnout. I přesto je pak čip schopný v reálném čase přijímat MP3 stream a dekódovat ho. V tuto chvíli je příjem náchylnější na jitter ve spojení, vše však probíhá

hladce i bez větší vyrovnávací paměti. Dokonce ani DAC není nezbytně nutný. Kód obsahuje možnost využití I2S rozhraní k účelu, ke kterému nebyl navržen – jako 5-ti bitový PWM generátor. Toto řešení (oproti klasickému PWM) s sebou nese výraznou úsporu procesorového času. Tak se zároveň sníží kvalita poslechu. ESP2866 také není navrhovaný pro dodávání signálu do induktivní zátěže. Právě poslední dva zmíněné fakty jsou hlavním důvodem, proč toto zapojení raději nejdříve vyzkoušet. Každopádně zmíněné úpravy znatelně sníží i cenu celého setu a komplikovanost zapojení.

Po kompilaci modifikovaného programu a jeho nahrání do ESP-12E modulu pomocí USB – UART převodníku jsem přes RC člen (dolní propust) připojil sluchátka na GPIO3 (I2S). Zvuk byl překvapivě dobrý. Reprodukce se ještě o něco zlepšila přidáním vyhlazovacího elektrolytického kondenzátoru mezi regulátor a ESP modul. U finálního výrobku bude potřeba o něco lépe odfiltrovat vyšší frekvence, jelikož přes takto jednoduchý článek proniká citelné množství šumu.

Experimentoval jsem též s delta-sigma modulací. Výsledek se mi zdál subjektivně horší než v předchozím případě. Tato modulace je navíc výpočetně mnohem náročnější, než dříve použitá PWM. Základní takt procesoru proto nestačí, je nutné použít dvojnásobný, tedy 160 MHz. To s sebou nese větší nároky na napájení, chlazení a také se podepisuje na výdrži baterie. Tuto možnost jsem proto zavrhl.

2.4.2 Napájení

Úbytek na regulátoru napětí je (během přijímání a reprodukce streamu) 0,3 V. (Údaj z katalogového listu přitom hovoří o 0,09V [6].) Při napájení z li-ion článku může být jeho teoretické konečné napětí 2,8 V. Ze zdroje testovaný obvod odebíral průměrně 70 mA. Postupně jsem zkoušel chování modulu od 4,3 V a napětí pomalu snižoval. Kolem 3 V (měřeno na vstupu stabilizátoru) v reprodukovaném zvuku znatelně stoupá šum. Kritické vstupní napětí je 2,7 voltu, kdy již ESP občas zamrzá. To odpovídá 2,4 V na ESP modulu, což je již pod jeho dovoleným provozním napětím. Přitom už při napětí 3 V můžeme li-ion článek považovat za vybitý. Rozsah dovolených napětí celého zapojení je tak zcela dostačující.

Při napájení z reálné li-ion baterie jsem oproti laboratorním zdroji nezaznamenal žádnou změnu v kvalitě reprodukovaného zvuku ani v chování čipu. Jako zdroj dobře posloužila vysloužilá baterie Samsug EB615288VU s nominálním napětím 3,7V a kapacitou 2500 mAh.

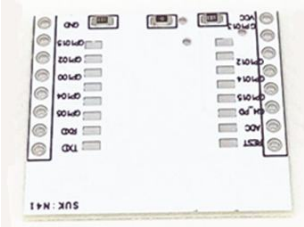
2.4.3 Ostatní HW

Jak je patrné z obrázku 2.1, celé zapojení je velmi jednoduché, realizované na kontaktním poli.

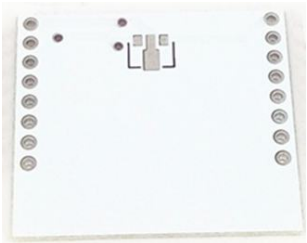
Filtrem je jednoduchá RC dolní propust. K filtru samotnému se ještě vrátíme v sekci 3.5.3.

Použitý modul má označení ESP-12E. Protože rozteč pinů není kompatibilní s kontaktním polem, použil jsem PCB adaptér pro ESP-12 (viz obr. 2.2), na který jsem modul naletoval. Příslušný adaptér se dá sehnat na oblíbeném tržišti `www.aliexpress.com` (v přepočtu kolem 10 Kč/kus).

Pozornému čtenáři jistě neunikl fakt, že název modulu obsahuje oproti určení adaptéru navíc písmeno „E“. To v tomto případě značí, že modul má oproti svému předchůdci vyvedeno více pinů. V našem případě to ale nevadí, jelikož chybějící piny nebudeme potřebovat. Přebývající vývody nikterak nepřekáží, protože jsou zhotovené formou prokovených otvorů na okraji destičky.



Obrázek 2.2. Adaptér pro ESP-12 - vrchní strana

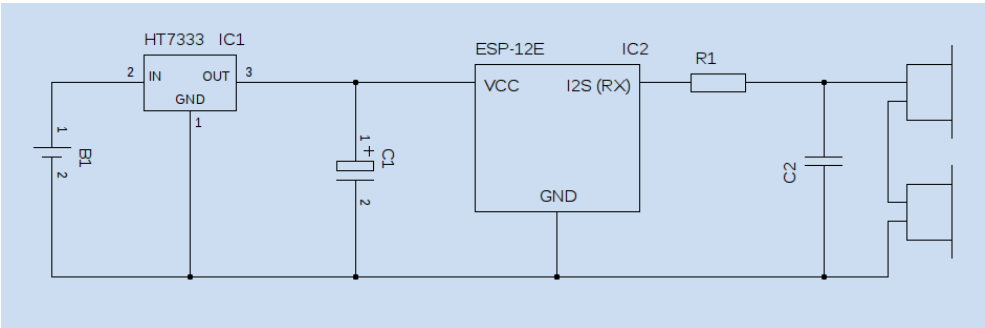


Obrázek 2.3. Spodní strana adaptéru

Zmiňovaný adaptér už má mimo jiné na své spodní straně (obr. 2.3) připravené pájecí plošky pro osazení regulátoru napětí. Po jeho připojení pak stačí odpájet příslušný zkratovací rezistor na druhé straně destičky.

Další nezbytnou součástí jsou sluchátka. Já jsem použil Canyon CNE-CHP2 s impedancí 32 Ω.

Celé zapojení je patrné ze schématu 2.4. Elektrolytický kondenzátor C1 kapacitou 100 μF pokrývá proudové špičky při vysílání. Bez jeho přítomnosti je ESP nadále stabilní, avšak pokles kvality reprodukce je již znát. Adaptér bohužel neumožňuje rozumnou možnost připojení blokovacích kondenzátorů poblíž vývodů regulátoru tak, aby bylo vyhověno doporučenému zapojení.



Obrázek 2.4. Schéma testovacího zapojení

2.4.4 Zhodnocení

Kvalitu reprodukovaného zvuku jsem pro daný účel subjektivně vyhodnotil jako dostačující. Celé zapojení také vyhovělo požadavkům na stabilitu.

Kapitola 3
Návrh HW přijímače

3.1 Program

3.1.1 SDK pro C

První pokusy o napsání programu směřovaly k využití jazyka C. Jako nevhodnější SDK se jeví esp-open-sdk [10], které je tak otevřené, jak jen bylo možné.

Kromě samotného NONOS SDK [7] obsahuje implementaci dalších projektů. Z těch stojí za zmínku kupříkladu lwIP, který má velmi dobrou dokumentaci [11].

Kromě NONOS SDK Espressif také vyvinula RTOS SDK, které se liší svou základní filozofií. Zatímco v NONOS se používají ke spouštění příkazů časovače a callback funkce, RTOS SDK je orientované na procesy, jejich komunikaci a synchronizaci, multitasking. Program se pak jeví, jako by byl určen spíše pro operační systém, než mikročip.

SDK od Espressif sice také obsahuje funkce pro vytvoření socketu, jejich použití je ovšem limitované a ne zcela přímočaré. Zrovna toto SDK je nepostradatelnou součástí, jelikož nízkoúrovňová rozhraní jsou obchodním tajemství firmy.

Po instalaci esp-open-sdk se ukázalo, že učit se programovat v C na takto specifickému kusu HW je velice náročné. Příklady použití LwIP a ani obecné příklady použití IP stacku v C se mi nepodařilo, i přes zevrubné studium obou SDK a syntaxe jazyka C, zprovoznit.

Mezi hlavní důvody pro využití jazyka C patří:

- pružnost výsledného kódu
- rychlý vývojový cyklus
- efektivní využití procesorového času
- efektivní využití RAM

Pružnost je vcelku jasná. Vzhledem k tomu, že C je poměrně nízkoúrovňový jazyk, dají se některé záležitosti řešit mnohem elegantněji, než ve vyšších jazycích. Rychlým vývojovým cyklem je myšlen krátký čas mezi provedení změny v kódu a reálným testem přeloženého programu.

SDK nejsou apriori závislá na použití konkrétního operačního systému. Pod Linuxem se jedná o stažení potřebných souborů z Githubu, kompilaci a následnou instalaci. Pokud se ovšem rozhodnete pro OS Windows, budete řešit celou řadu dílčích problémů. Dokonce i Espressif (a fanoušci ESP) doporučují pod tímto systémem nainstalovat virtuální Linuxový počítač, což je bohužel v tomto případě nejschůdnější varianta. Firma též nabízí řadu návodů a již předpřipravené diskové obrazy s instalovanými nástroji, které stačí pouze spustit [12].

3.1.2 Wiring

Jde o objektově orientovaný programovací jazyk platformy Arduino. Se všemi výhodami a nevýhodami, které z tohoto faktu plynou. Přeložení programu trvá výrazně delší dobu.

Výsledný binární soubor je také větší. Tedy i jeho nahrání do flash paměti tedy trvá delší dobu. Výhodou je poměrně jednoduchá syntaxe. Na rozdíl od C se pak programátor nemusí starat o spoustu věcí, jako je přidělování a uvolňování paměti, studium konkrétního SDK pro konkrétní čip a podobně.

Arduino IDE používá také LwIP, avšak je nad ním vybudovaný systém objektů. Hlavní výhodou potom byl fakt, že se jedná o velmi oblíbené prostředí a přímo pro ESP8266 existuje celá řada příkladů použití. Ty se ovšem (na rozdíl od dokumentace LwIP) dají dohledat jako samostatné funkční celky, ze kterých se dá snadno pochopit princip a velmi jednoduše se s nimi experimentuje. Proto jsou vhodné i pro lidi s minimálními programátorskými zkušenostmi.

3.1.3 Instalace prostředí Arduino IDE

Instalace je oproti SDK pro jazyk C opravdu velmi jednoduchá. Jedná se o italský projekt, jehož původním účelem je výuka programování na školách. Vývojové prostředí je napsané v Javě, takže je nezávislý na konkrétní platformě. Existuje tak ve verzi pro Linux, Mac OS ale dokonce i Windows.

Stačí si ze stránek projektu stáhnout příslušný instalační balíček [13], případně nainstalovat z repozitáře vašeho OS. Dále je potřeba mít nainstalované ovladače pro použité sériové převodníky (to samozřejmě platí i pro ostatní způsoby vytváření firmware pro ESP). A v poslední řadě, nainstalovat podporu pro vývojové desky s ESP [14].

Zmíněnou podporu přidáte tak, že (v české verzi) otevřete nabídku *Soubor* → *Vlastnosti* a v kolonce *Správce dalších desek URL* přidáte řetězec:

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

V případě, že uvedené pole již nějakou adresu obsahuje, jednotlivé URL je potřeba oddělit čárkou. Názornou ukázku naleznete na obrázku 3.1.

Poté je již možné desku nainstalovat. Zvolíme *Nástroje* → *Vývojová deska* → *Manažér Desek*. Vyskočí nám okno jako na obrázku 3.2. Poté stačí vyhledat příslušnou desku. V našem případě se balíček jmenuje *esp8266*. Vybereme verzi, kterou chceme nainstalovat a klikneme na *Instalace*, čímž se celý proces spustí.

3.2 Komunikační protokol

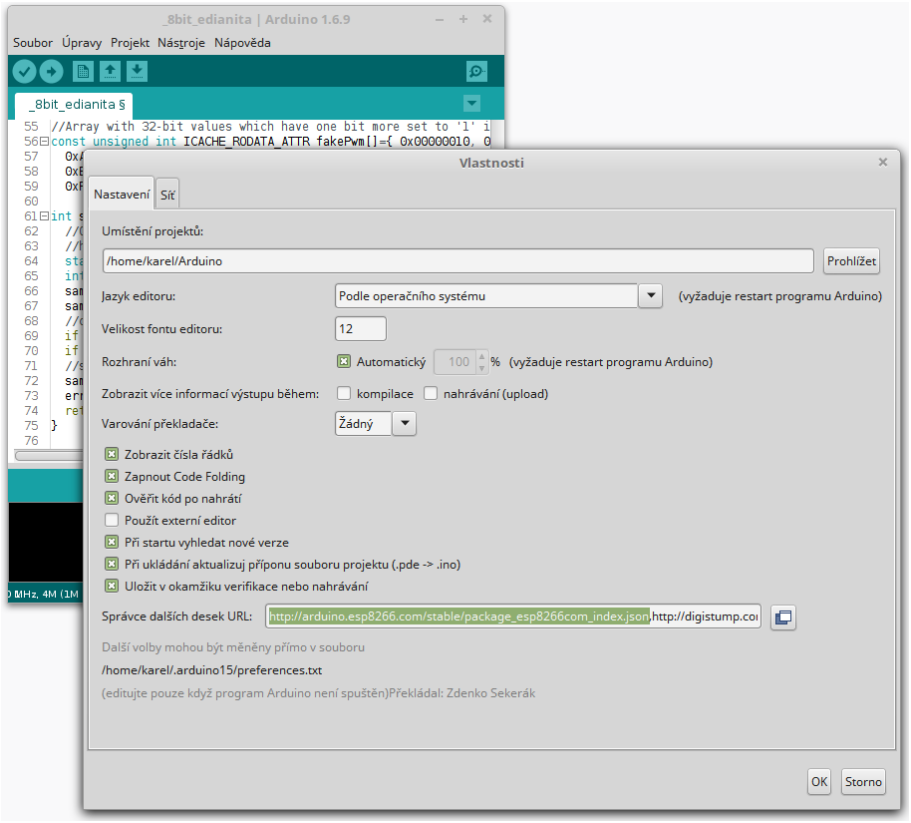
Tato sekce se bude zabývat především strukturou dat, která si mezi sebou budou připojená zařízení vyměňovat.

3.2.1 Signalizace

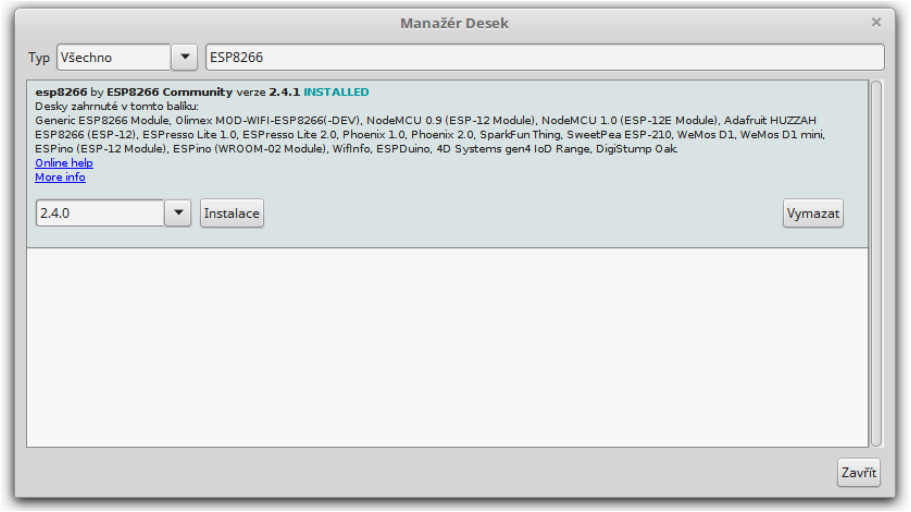
Jedním z požadavků na toto zařízení je nutnost detekce ztráty klienta a sledování stisku tísňového tlačítka. To vyplývá přímo ze zadání. Jelikož mohou být na hotspot průvodce připojena (kromě těchto HW přijímačů) i jiná zařízení, není možné ztrátu klienta detekovat tím, že se stanice odpojila od WiFi.

Z tohoto důvodu jsem se rozhodl, že o sobě budou stanice dávat v pravidelném intervalu vědět pomocí krátkého UDP paketu. Každá stanice má unikátní ID, které získává během nahrávání firmware. Toto ID je součástí všech odesílaných paketů, aby bylo jasné, od které stanice pochází.

Ve snaze o co nejjednodušší návrh není komunikace nikterak zabezpečena. Je však dána pevná délka datagramu, která činí 3 bajty. Již toto opatření samo o sobě vyloučí náhodně přijaté pakety na daný port, protože ty budou s největší pravděpodobností delší. Dalším opatřením je fakt, že veškeré zprávy mají první bajt nastavený na (při



Obrázek 3.1. Ukázka přidání podpory zařízení v prostředí Arduino IDE



Obrázek 3.2. Ukázka přidání desky v manažéru desek prostředí Arduino IDE

psaní kódu náhodně zvolenou ale pevnou) hodnotu `0xFA` (hexadecimálním zápis). Náhodný UDP paket tedy tuto komunikaci nenaruší, jelikož je velmi nízká pravděpodobnost, že by doputoval na správnou IP adresu, správný port, měl přesně 3 bajty a ještě k tomu začínal na `0xFA`.

Ve druhém bajtu je potom uloženo ID stanice, která tento paket odeslala. Třetí bajt obsahuje typ zprávy. Tím je buď již zmíněná periodická odpověď s hodnotou `0x00` a pak reakce na stisknutí tlačítka s `0x01`.

Všechny tyto zprávy se posílají na adresu výchozí brány, což je v tomto případě adresa samotného telefonu, pro který jsou tyto zprávy určeny. Paket se posílá na port 4445.

3.2.2 Příjem zvuku

Zařízení naslouchá na portu 4210 a očekává UDP stream obsahující čistá data kódovaná pomocí PCM. Naslouchání probíhá na všech IP adresách zařízení, to jest IP adrese přidělené z DHCP, broadcastové IP adrese sítě a zároveň i na multicastové `239.0.0.57`. Poslední jmenovaná adresa je pro příjem nejvhodnější. Jelikož datový tok je stálý a poměrně velký, při unicastovém vysílání by s rostoucím počtem přijímajících stanic úměrně rostl datový tok. Broadcast je oproti tomu posílán i stanicím, které o tato data nemají zájem. Multicastové pakety dorazí jen stanicím, které jsou přihlášeny k odběru a ostatní tím nejsou zatěžovány. Přitom serveru stačí každý paket vyslat jen jednou, jako v případě broadcastu, takže není zbytečně přetěžován zdroj dat, ani stanice, které o tento druh vysílání nestojí.

PCM kódování jsem použil pro svou jednoduchost. ESP má sice poměrně velký výkon, s paměti RAM je však potřeba šetřit. Složité dekódování přijatého signálu kvůli úspoře datového toku tak není vhodné.

3.3 Zpracování signálu

Vzhledem k jednoduchosti kódování se dá celé zpracování signálu provést v následujících dvou krocích.

3.3.1 Uspořádání dat

Přijatá data získáme jako bytový proud. Je tak nutné jednotlivé byty uspořádat do proměnné, která bude reprezentovat konkrétní vzorek. Záleží na daném bitovém rozlišení signálu, tedy zda-li přijímáme 8, 16 nebo 32 bitovou PCM modulaci. Dalším faktorem je počet kanálů, tedy jestli se jedná o mono či stereo signál, a tudíž jeden nebo dva proudy. V případě příjmu dvou kanálu se vzorky pro levý a pravý kanál periodicky střídají.

Při vyšším rozlišení (16 nebo 32 bitů) je též nutné uvažovat edianitu. Jedná se o fakt, že v tomto případě zabírá jeden vzorek (pro každý kanál) více bitů. Jsou ovšem dvě varianty přenosu. V jedné se přenáší méně významné byty jako první (tzv. little-endian) a ve druhé se začíná od nejvýznamnějšího bytu (big-endian).

3.3.2 Reprodukce

Pro Arduino existuje knihovna starající se o zvukový výstup – i2s.h [15]. Při použití vhodného I2S DAC dostaneme přímo audio signál v maximální kvalitě. Struktura dat pro komunikaci po I2S je velmi dobře popsána v tomto zdroji [16].

Z uvedeného také plyne, že velikost proměnné, kterou knihovna očekává je 32 bitů a obsahuje data pro levý a pravý kanál. V našem případě bude stačit 16 bitů stereo, avšak pro jednodušší implementaci jsem velikost proměnné zachoval.

Jelikož jsem si již dříve ověřil, že kvalita PWM výstupu bude dostatečná (viz. 2.4.1), dalším krokem bylo přepočítání vzorků na PWM signál s tím, že jsem využil triků z MP3 dekodéru [9] – tedy konkrétně funkci `sampToI2sPwm()` na přepočet PWM jednotlivých vzorků do průběhu PWM. Odesílat tyto 32 bitové vzorky jeden po druhém by stálo velké množství procesorového času a již by nezbýval výkon na obsluhu UDP socketů. Inspiroval jsem se tedy i v případě výsledného odesílání vzorků a fyzickou I2S sběrnici

použil jako posuvný registr. Stačí do ní zapsat přepočítaný vzorek a o vlastní výstup se postará HW řadič.

Uvedená funkce z každého vzorku vybere 5 nejvyšších bitů, které nahradí 32 bitovým slovem. To už reprezentuje průběh potřebného PWM signálu.

3.4 Ladění programu

Vzhledem k tomu, že ještě nemáme zdroj zvukového proudu, ale přesto by byl takový zdroj velmi vítaným pomocníkem pro průběžné ladění části programu starající se o zvukový výstup, rozhodl jsem se použít program Ffmpeg, což je vynikající nástroj pro všemožnou konverzi audio a video formátů a zpracování obrazu a zvuku [17]. Požadovaný zvukový výstup získáme po zadání příkazu:

```
$ ffmpeg -re -i test.mp3 -f s16le -acodec pcm_s16le -ac 1 -ar 22050 -af
volume=2 udp://192.168.43.255:4210?broadcast=1
```

Tento řádek znamená, že chci:

- zdroj číst programem Ffmepg jeho nativní rychlostí (`ffmpeg -re`)
- data získávat ze zvukového souboru test.mp3 (`-i test.mp3`)
- enkódovat ho jako 16-ti bitový zvukový stream (`-f s16le -acodec pcm_s16le`)
- s jednou zvukovou stopou (`-ac 1`)
- a vzorkovací frekvencí 22050 Hz (`-ar 22050`)
- dálepak nastavit hlasitost 2 (`-af volume=2`)
- a nakonec výsledek odesílat jako UDP stream na uvedenou broadcastovou adresu (`udp://192.168.43.255:4210?broadcast=1`)

Varianta s little-endian přesně odpovídá kódování, které používá OS Android, takže v pozdější verzi nebude potřeba nic měnit. Díky tomu, že se dá poměrně jednoduše odesílat zvuk z počítače jsou přijímače použitelné kupříkladu s miniaturními počítači, jako je třeba Raspberry Pi.

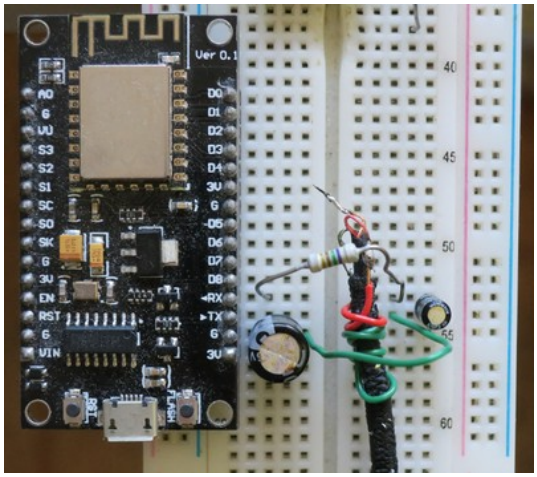
3.5 Fyzická realizace

3.5.1 NodeMCU

Vzhledem k tomu, že problematikou testování jsme se zabývali již v sekci 2.4, můžeme se spolehnout na to, že výsledné zapojení bude funkční i se samostatným modulem. Pro vývoj a ladění jsem použil desku NodeMCU. Vývojové prostředí Arduino IDE si dokáže automaticky provést resetování desky do režimu pro flashování, nahrát nový FW a modul znovu resetovat pro běh nově nahraného programu. Kompilace a nahrání FW je tak otázkou jednoho kliknutí.

Mezi piny sériové linky ESP a převodníku USB na UART je u této vývojové desky zařazen rezistor o hodnotě 470 Ω. Bez něj by pro náš účel nebyla použitelná, protože RX pin na ESP je zároveň pinem pro I2S. Takže v tomto případě jednak zvukovým výstupem a současně slouží pro příjem dat sériové linky – tedy především nahrání nového programu.

Nyní to ovšem nevadí, protože pokud jsou v jednu chvíli oba piny nastavené jako výstup, rezistor se postará o omezení proudu na bezpečnou úroveň a tak není potřeba mít strach o zničení ani jedné ze součástek.



Obrázek 3.3. Ukázka zapojení použitého během vývoje a ladění aplikace pro ESP8266

Celé zapojení použité při vývoji si můžete prohlédnout na obrázku 3.3. Je zde vidět samotná vývojová deska, která obsahuje už vše potřebné, jako například převodník USB – UART CH340, lineární stabilizátor napětí AMS1117 a samozřejmě také modul ESP-12N. Napájení NodeMCU jsem podpořil kondenzátorem 470 μF. I zde vidíte jednoduchý filtr, více popsaný v 3.5.3, a na spodní straně připojený kabel od sluchátek.

NodeMCU disponuje dvěma tlačítky, která naleznete v blízkosti USB portu. Jsou jimi *Reset*, které slouží k resetování ESP a potom *Flash*. Pokud ho podržíte během nabíhání ESP (a je už jedno, zda se tak děje kvůli tomu, že bylo právě zapnuto napájení nebo byl čip resetován), pak čip očekává nahrání nového firmware přes UART. Tato tlačítka nejsou nezbytně nutná, protože NodeMCU umí reset i přepnutí do UART režimu provést prostřednictvím převodníku, přesto jsou situace, kdy se HW tlačítka hodí.

3.5.2 Ostatní moduly

S realizací na samostatných modulech je (oproti NodeMCU) o něco více práce, avšak ne o mnoho. Je potřeba pohlídat si logické úrovně při resetu a zapínání na určitých pinech (již sem se tím zabýval v kapitole 2.1.6). Také je nutné doplnit stabilizátor napětí na 3.3 V a samozřejmě též nezbytné kondenzátory.

Pro napájení z baterie bude použit nabíjecí modul s TP4056 s ochranou proti podpětí.

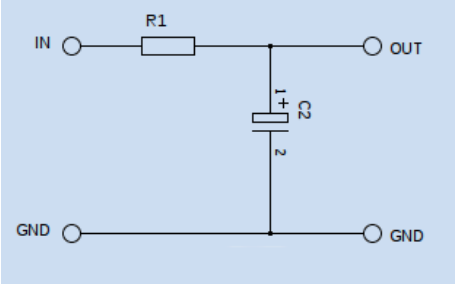
3.5.3 Filtr

Sluchátka nelze připojit přímo na ESP hned ze dvou důvodů. Tím závažnějším je fakt, že výstupy ESP nejsou uzpůsobeny pro dodávání proudu do induktivní zátěže. Velmi pravděpodobně by některá ze vzniklých napěťových špiček modul zničila. Druhým důvodem je skutečnost, že surový PWM signál se mimo jiné skládá z vysokých kmitočtů, které by kazily výslednou reprodukci.

Proto je nepostradatelné zařazení filtru. Ten může být pro tento účel opravdu jednoduchý, postačí RC článek (konkrétněji dolní propust). Hodnoty 1 kΩ a 100 nF [18] se ukázaly jako vyhovující.

Empiricky jsem zjistil, že se zvětšením kapacity kondenzátorem ubývá ruchů na pozadí. Jak se však dá očekávat, výsledná hlasitost reprodukce se tímto zásahem také snižuje.

Zapojení filtru je patrné z obrázku 3.4



Obrázek 3.4. Schéma použitého filtru - dolní propust

3.6 Popis funkce

Kvůli předpokládanému použití laiky (turisty), byl brán zřetel na maximální jednoduchost použití. Výsledný produkt proto bude mít jen dva ovládací prvky – vypínač napájení a tísňové tlačítko. Po zapnutí automaticky vyhledá přednastavené AP a připojí se k němu. SDK počítá také s automatickým znovupřipojením v případě ztráty spojení.

Po zapnutí začne přístroj naslouchat na portu 4210 a v případě, že je aktivní vysílání, reprodukovat přijatý zvuk. Nastavení hlasitosti se děje na straně mobilního telefonu.

Při libovolně dlouhém stisku tlačítka se do mobilního telefonu odešle zpráva s tísňovou žádostí.

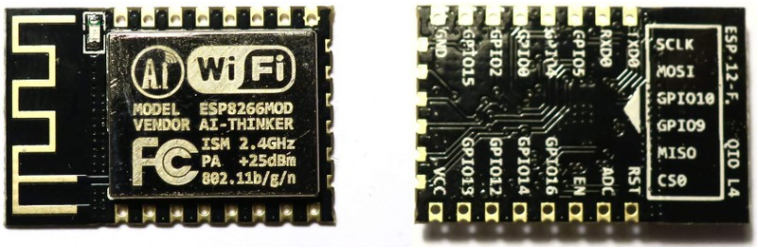
Kapitola 4
Prototyp

4.1 Výroba

Cílem výroby dalšího prototypu je udělat generální zkoušku všech komponent tak, jak je s nimi uvažováno ve finálním produktu. Tedy včetně relativně malé baterie, nabíjecího modulu, vypínače, tlačítka, regulátoru a levného sluchátka.

4.1.1 Modul ESP-12F

Po odladění programu a testech jsem přistoupil k výrobě druhého prototypu, tentokrát již v podobě kompaktního zařízení. Srdcem je modul ESP-12F který můžete vidět na obrázku 4.1).



Obrázek 4.1. Modul ESP-12E – pohled na horní (vlevo) a spodní stranu

Jeho hlavní výhodou, oproti většině ostatních modulů, je velký počet pinů, kvalitnější meandrová anténa, nízká cena a indikační LED dioda připojená k pinu GPIO2.

Veškerá elektronika je schovaná pod kovovým krytem, který zabraňuje pronikání rušení směrem dovnitř, ale taktéž únikům od interní elektroniky ven.

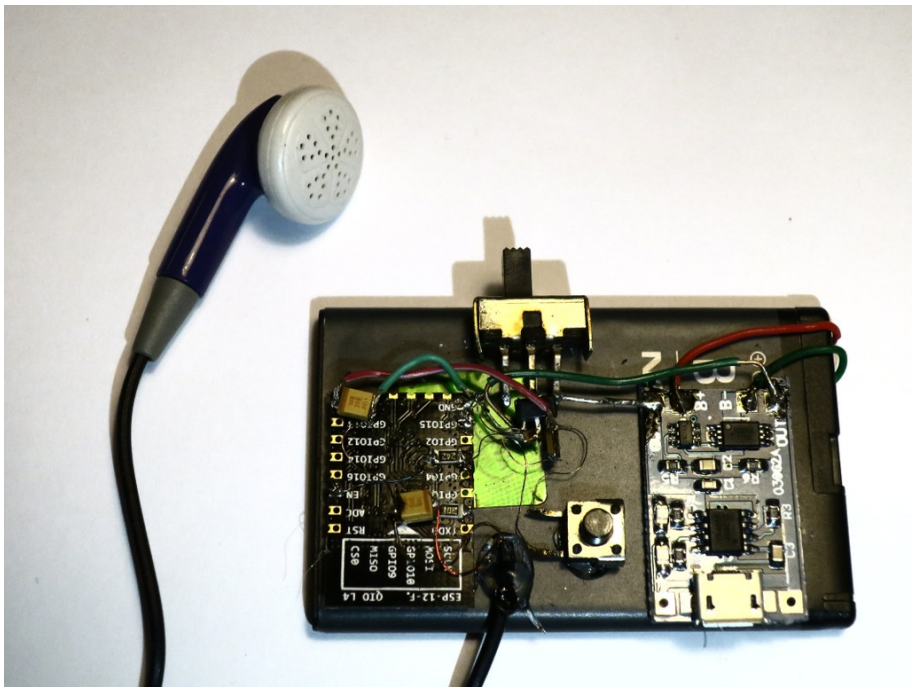
4.1.2 Prostorové rozvržení

Vzhledem k absenci PCB, se kterým je počítáno pro finální produkt, započal jsem výrobu jako „vrabčí hnízdo“, tedy spojováním součástek především pomocí jejich vlastních vývodů. Protože velké množství použitých komponent je bezvývodových, úměrně přibylo drátových propojek. U obvodu této složitosti tato skutečnost ovšem nehraje podstatnou roli.

Jako hlavní konstrukční prvek posloužila baterie Nokia BL-4J s reálně změřenou kapacitou 800 mAh. Kapacitu jsem změřil při nabíjení pomocí USB měřidla KEWEISI s tolerancí ±5 %. Přesnost není nikterak oslnivá ale pro naše účely postačuje. Nabíjecí elektronika modulu neobsahuje žádný konvertor, takže vstupní proud zhruba odpovídá proudu na výstupu. Baterii jsem vybral kvůli svým vhodným rozměrům, dostatečné kapacitě a také kontaktům. Mezi ty se dá vsunout drát, a tak se i bez pájení připojit na její kladný a záporný pól. Detail kontaktů můžete vidět na obrázku 4.2.

Na baterii jsem všechny ostatní komponenty připevnil pomocí hmoty z tavné pistole. Není to ideální druh spojování předmětů, avšak má tu výhodu, že pokud se později

Obrázek 4.2. Detail kontaktů baterie



Obrázek 4.3. Fotografie prototypu

rozhodnete některou ze součástek vyměnit, stačí ji pouze šetrně odtrhnout. Jednoduché změně návrhu nahrává i samotná konstrukce spojů. Provést úpravu v zapojení je tak dílem okamžiku. Celkový přehled si uděláte, pokud se podíváte na obrázek 4.3.

Vstupní pin regulátoru HT7333 je prostřední, který zároveň slouží i k odvodu tepla. Toho jsem s výhodou využil a naletoval ho přímo na pin přepínače. Přidal jsem dva kondenzátory těsně k vývodům regulátoru. Pro vyhlazení proudových špiček jsem na napájecí piny modulu přidal tantalový kondenzátor 10 μF.

Celý modul jsem se rozhodl umístit horní stranou dolů z několika důvodů. Předně jsem nechtěl, aby byla anténa příliš vystrčená mimo baterii. Tímto vznikl mezi baterií a anténou alespoň minimální odstup. Popisky pinů jsou díky tomu dobře čitelné a zapojování tak bylo snadnější. Z druhé strany navíc není kvůli rozměrnému stínění v okolí pinů téměř žádný prostor. Kyt je také zhotoven z vodivého materiálu, zatímco zadní strana je pokryta nepájivou maskou, která zamezí zbytečným zkratům.

4.1.3 Drátové propojky

Propojky mezi baterií a nabíjecím modulem jsou zhotoveny z plného měděného drátu, aby dobře držely zaklesnuté v kontaktech. Ostatní napájecí vodiče jsou ohebná lanka. Použití drátu tady není možné, protože letovací plošky by se mohly při silnějším mechanickém namáhání snadno odtrhnout a modul by tak byl znehodnocen.

Signálové vodiče jsou z tenkého lakovaného (transformátorového) drátku. Je jím připojen pull-up rezistor pinu GPIO0 a pin EN (označovaný také CH_PD) na VCC a dále

pak tlačítko. To je připojeno mezi záporný pól a GPIO0, ke kterému se ještě vrátíme ve 4.1.4.

Jak vyplývá z tabulky 2.1, je také potřeba zajistit, aby pin GPIO15 byl při startu v logické nule. I když tento pin nevyužíváme, propojení pevnou drátovou propojkou by nebylo příliš vhodné, jelikož by vinou chyby programátora mohlo dojít ke zničení mikrokontroléru. To by nastalo případě, že by program přepnul pin na výstupní a snažil se vynutit logickou jedničku.

Již jsem viděl několik funkčních zapojení, kde byly piny ovlivňující běhový mód napevno připojené ke kladnému nebo zápornému pólu napájení. Pokud budete během bootování sledovat poblikávání LED diody připojené k pinu 2, stejně jako mne vás možná napadne otázka, zda-li se takto nechovají i ostatní piny zodpovědné za výběr módu zavádění programu. Pin EN (CH_PD) je pouze výstup, tedy tam žádné riziko při přímém připojení nehrozí.

Pin označovaný jako RST naopak můžeme nechat zcela bez povšimnutí. Jedná se o negovaný reset (logická nula reset aktivuje). Interní pull-up ho udržuje na úrovni logické jedničky, takže pokud k pinu nic nepřipojíme, ESP naběhne v požadovaném módu.

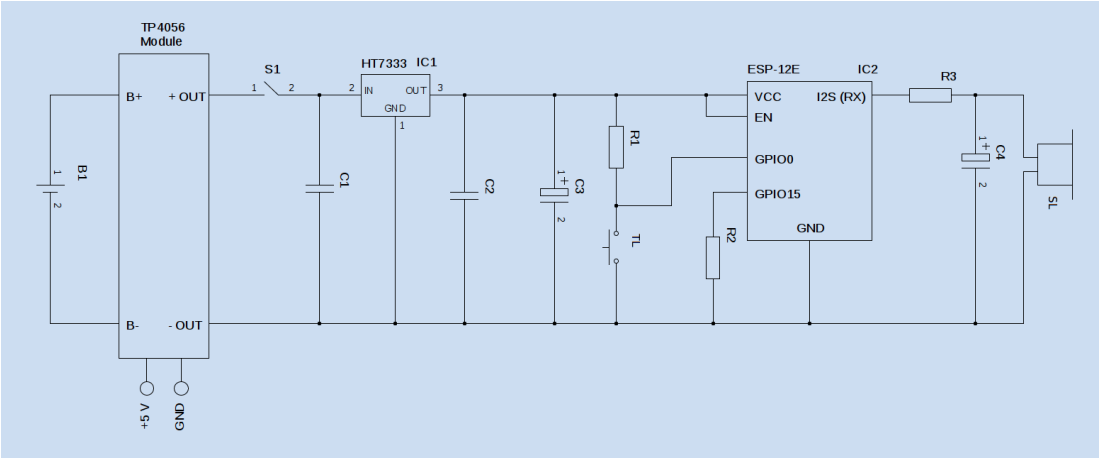
4.1.4 Tlačítko

Zapojením tlačítka jsem se inspiroval u vývojové desky NodeMCU, kde je tlačítko *Flash* připojené na GPIO0. Tam slouží k přepnutí modulu do UART režimu. Zároveň ho však lze využít i jako vstup při běhu vlastního kódu.

Potřebujeme mít alespoň jedno tísňové tlačítko pro uživatele tohoto přijímače a zároveň se může hodit možnost snadného nahrání nového firmware. Tímto způsobem může obě funkce zastat jediné tlačítko.

4.1.5 Schéma

Na obrázku 4.4 najdete celkové schéma zapojení. Jak je patrné, příliš se neliší od testovacího přípravku na nepájivém poli (Schéma 2.4).



Obrázek 4.4. Celkové schéma zapojení prototypu

Hodnoty jednotlivých použitých součástek naleznete v tabulce 4.1.

Součástka	Označení	Hodnota
Baterie	B1	800 mAh
Spnač	S1	–
Tlačítko	TL	–
Lineární stabilizátor	IC1	HT7333 [6]
Modul ESP12E	IC2	–
Napájecí modul	TP4056	–
Sluchátko	SL	8 Ω
Kondenzátory	C1, C2	50 nF
	C3	10 μF
	C4	1 μF
Rezistory	R1, R2	1 kΩ
	R3	330 Ω

Tabulka 4.1. Tabulka použitých součástek

4.2 Měření

4.2.1 Výdrž na baterii

Očekávaná výdrž modulu s touto baterií byla právě kolem osmi hodin. Vycházel jsem přitom z přibližné spotřeby 80 mA [19–20]. Prostým podělení kapacity baterie (800 mAh) spotřebou dostaneme údaj o něco vyšší, avšak další aditivní zátěží byla reprodukce do sluchátka a svit stavové LED diody. Také je nutné počítat s tím, že se kapacita baterie nevyužije maximálním způsobem a pár procent v ní zůstane i po vypnutí zařízení.

S plně nabitou baterií a poslechem až do úplného vybití reprodukce trvala 6 hodin a 26 minut. Prostou lineární aproximací tedy zjistíme, že požadované výdrže 8 hodin (viz zadání této práce) lze dosáhnout s baterií 990 mAh.

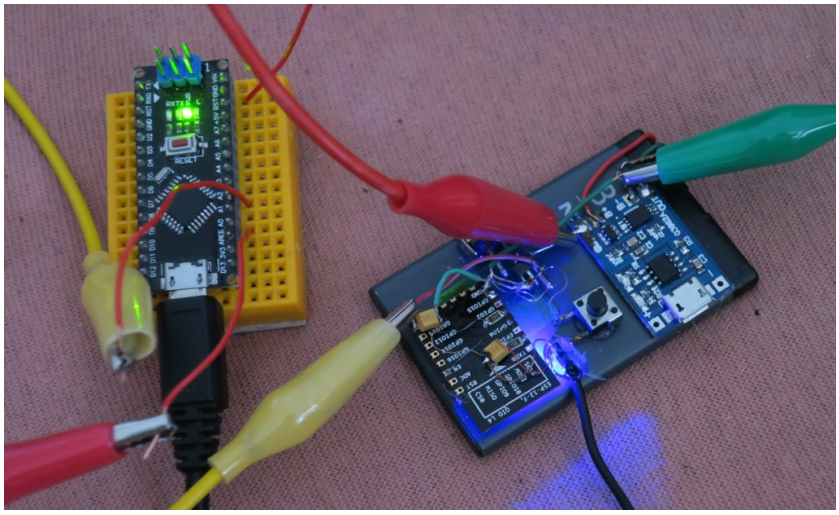
Pro lepší představu o průběhu pokusu jsem měření na baterii a zároveň výstupu regulátoru prováděl periodicky v intervalu 1 sekunda. Údaje získané v průběhu uplynulé minuty zprůměroval a po sériové lince odeslal k zaznamenání do počítače. Princip záznamu je tak velmi podobný tomu použitému v mé bakalářské práci [21], kde se ovšem zaznamenává větší množství údajů, jako například proud, vnitřní odpor zdroje a podobně. Přístroj však disponuje pouze jedním kanálem pro každou veličinu, proto jsem si napsal krátký skript pro Arduino, který obstaral vše potřebné.

Situační fotografii naleznete na obrázku 4.5. Jak je patrné, nebylo nutné použít napěťových děličů, Arduino lze využít do 5 V jako napěťovou sondu. Stačí tedy přímo připojit k měřenému napětí a propojit země obou přístrojů. Je však potřeba dbát zvýšené opatrnosti, jelikož v tomto případě je zem společná zároveň i pro USB port. Měřený objekt je tak galvanicky přímo připojený k portu vašeho PC. Při rozdílu potenciálů hrozí poškození měřeného přístroje, USB portu počítače nebo samotného Arduina.

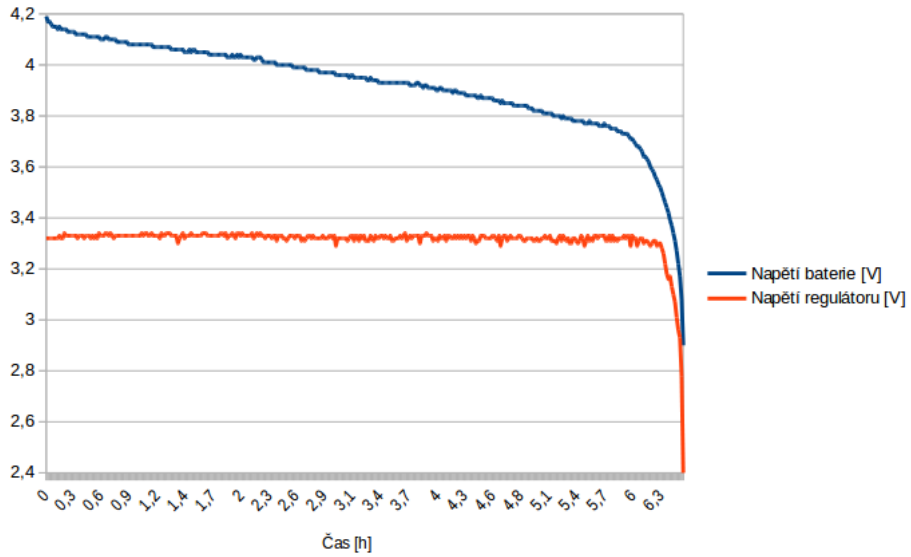
Na obrázku 4.6 můžete vidět vizualizaci naměřených dat. Přestože napětí baterie pomalu klesá, lineární stabilizátor téměř po celou dobu drží konstantní výstupní napětí 3,3 V. Je zde také dobře patrná typická charakteristika lithiového článku, kdy na začátku a konci vybíjení napětí strmě klesá a v mezilehlé oblasti je téměř lineární.

4.2.2 Charakteristika výstupního signálu

Abych zjistil zkreslení na výstupu, vygeneroval jsem si tři sinusové signály o frekvencích 440, 622 a 5 00 Hz. Ty jsem odvysílal pomocí Ffmpeg a výstup ze zařízení zase



Obrázek 4.5. Fotografie měření napětí pomocí Arduina

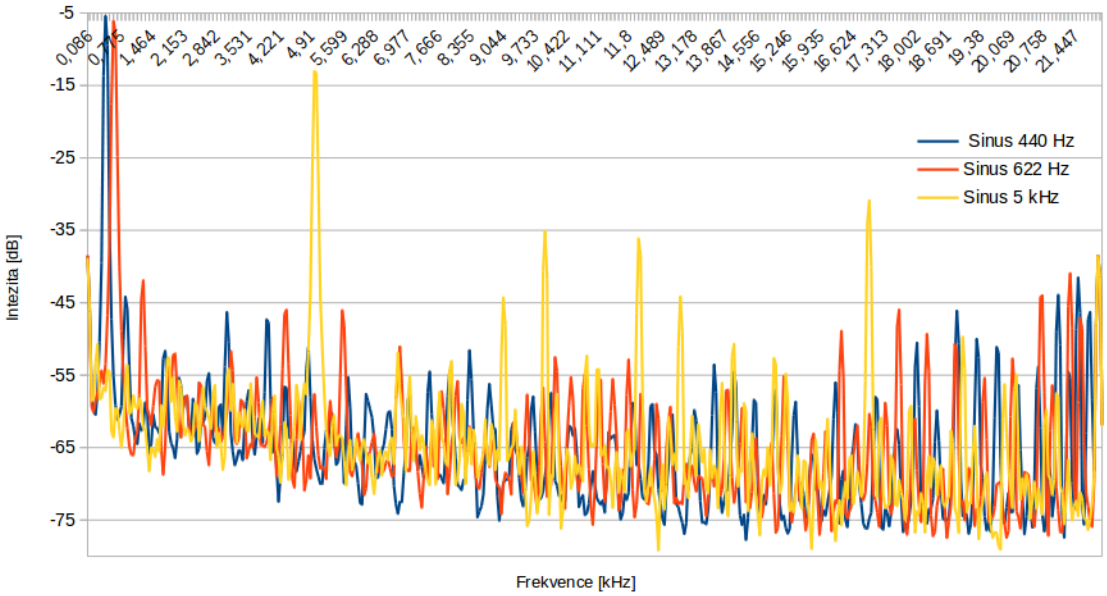


Obrázek 4.6. Graf závislosti napětí baterie na čase

nahrál. Tímto jsem dostal původní signál, který je navíc zkreslený průchodem celým řetězcem. Pomocí rychlé Fourierovy transformace jsem vypočítal spektrum těchto tří signálu a vynesl do grafu 4.7.

V ideálním případě bychom zde viděli pouze tři špičky, patřící každý jedné ze zmíněných frekvencí. Protože však došlo ke zkreslení signálu, přibyly tam i další frekvence. Nejlépe je to vidět na příkladu sinusovky o frekvenci 5 kHz, která má hlavní špičku s velikostí -10 dB a druhou nejvyšší -30 dB na frekvenci 17 kHz. V případě zbylých dvou zkoušených frekvencí je odstup signálu od šumu lepší, rozdíl nejvyšších dvou špiček činí 32 dB (= -5 dB -(-37 dB)).

Odfiltrováním vyšších kmitočtů lze výsledný dojem z reprodukce zlepšit, jelikož frekvence nad 10 kHz již nejsou pro srozumitelnost mluveného slova příliš podstatné.



Obrázek 4.7. Graf spektra signálu na výstupu přípravku

Kapitola 5
Mobilní aplikace.

5.1 Vývojové prostředí

Pro vývoj mobilní aplikace jsem zvolil prostředí Android Studio [22] společnosti Google. Samotné IDE dále obsahuje:

- Kompilátor
- Android SDK Tools
- Emulátor

V zásadě tedy vše, co je pro vývoj mobilní aplikace potřeba. Instalace tohoto balíku znamená rozbalení komprimovaného archivu a spuštění instalátoru.

Toto prostředí je velmi nápomocné. Rozložení prvků na obrazovce je zapsáno v XML souboru, nebo se dá „naklikat“. Prostředí zvýrazňuje syntaxi. Napovídá jak se dají konkrétní objekty, zobrazuje nepoužité knihovny a proměnné. Pokud nějaká funkce není dostupná v cílovém API, doporučí jeho povýšení.

Programátorovi s naprosto minimálními zkušenostmi (jakožto i autorovi této práce) zmíněná skutečnost velmi pomůže, jelikož má okamžitou zpětnou vazbu a ještě před kompilací ví, že program neobsahuje hrubou chybu. Ta přitom nemusí být jen syntaktická, může se jednat například o chybějící inicializaci proměnné před jejím použitím a podobně.

5.2 Popis jednotlivých funkcí

Aplikace určená pro průvodce disponuje jedinou stránkou, které se v Android studiu říká aktivita. Z této aktivity jsou dostupné veškeré funkce aplikace, průvodce tedy nemusí nic hledat v menu, vše důležité má uspořádáno před sebou na jedné obrazovce.

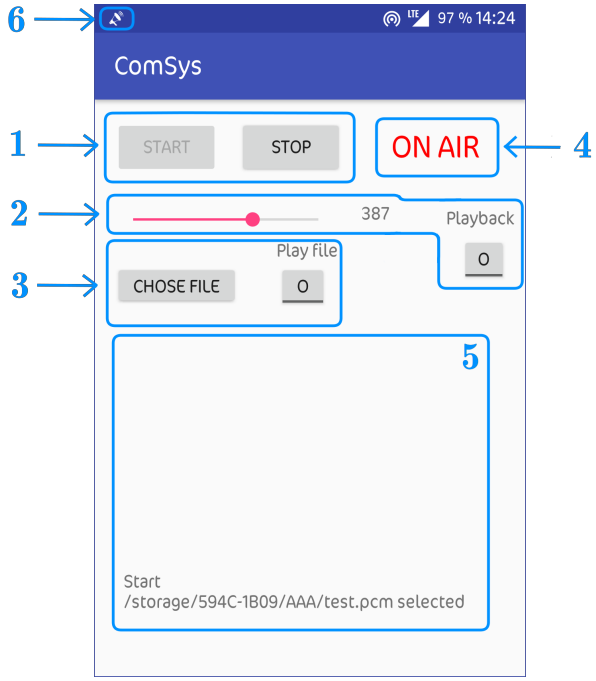
5.2.1 Tlačítka Start / Stop

Tlačítkem *Start* je spuštěno zachytávání zvuku z mikrofonu a zároveň vysílání datového proudu. Souběžně si aplikace od tohoto okamžiku začne tvořit databázi právě připojených zařízení. Pokud se kdykoliv během vysílání přidá nějaká stanice, bude přidána také do databáze. O případném předčasném odpojení jakékoliv z nich bude průvodce do 30-ti sekund informován. Tlačítkem *Stop* se veškerá činnost aplikace zastaví.

5.2.2 Nastavení hlasitosti

Nastavení hlasitosti se provádí pomocí posuvníku. Tažením se v reálném čase upravuje hlasitost vysílání. Pro dosažení optimálního nastavení hlasitosti bez přebuzení (přetečení PCM vzorků) může průvodce využít tlačítko *Playback*, které zapne přehrávání zachyceného zvuku.

Velmi doporučuji využít sluchátek s mikrofonem. V první řadě se tak vyhnete ozvěně způsobené zpětnou vazbou mezi reproduktorem mobilu a mikrofonem. Hlavně ale není nutné po celou dobu výkladu držet mobilní telefon blízko ústům. Při připojení sluchátek s mikrofonem je zvuk automaticky brán z mikrofonu na sluchátkách.



Obrázek 5.1. Ovládací prvky aplikace

- (1) tlačítka *Start* a *Stop* 5.2.1
- (2) ovládání hlasitosti 5.2.2
- (3) přehrávání souborů 5.2.3
- (4) stavový box 5.2.5
- (5) stavový řádek 5.2.6
- (6) oznámení 5.2.7

5.2.3 Přehrávání souborů

Průvodce má s touto soupravou též možnost přehrání předpřipraveného souboru. Tlačítkem *Choose file* se otevře souborový manažer. Po vybrání příslušného souboru je o úspěšnosti této operace uživatel informován takzvaným „toast“ oznámením a také ve stavovém řádku výpisem cesty k souboru (viz. 5.2.6).

Je nutné, aby souborový manažer vracel cestu a ne kupříkladu URI. O tom, že se výběr souboru nezdařil je uživatel také informován. V případě nekompatibility je možné provést výběr souboru jiným souborovým správcem.

Pokud se momentálně vysílá průvodcův hlas, tlačítkem *Play* se spustí přehrávání, přičemž zvuk z mikrofonu v tuto chvíli není snímán. Novým stiskem tlačítka je možné přehrávání zastavit. Reprodukce se zastaví automaticky i v případě, že je již přehrán celý soubor. Poté se aplikace vrátí do vysílacího režimu.

Stisknutí tlačítka *Stop* zapříčiní zastavení přehrávání a současně i hlasové vysílání. Přehrávaný soubor také zazní z reproduktoru mobilního telefonu, případně připojených sluchátek. Z důvodu možné synchronizace průvodce s nahrávkou to byl jeden z požadavků zadání této práce. Pokud si průvodce nepřeje poslouchat právě reprodukovanou nahrávku, stačí na svém přístroji ztišit hlasitost přehrávaných médií.

5.2.4 Příprava souborů

Aplikace předpokládá mono PCM soubory s 22050 vzorky za sekundu a 16-ti bity na vzorek, tedy stejný formát, jako se vysílá po síti. Tyto soubory je možné si vytvořit

předem, pomocí obdobného příkazu jako je uvedeno v sekci 3.4. Jediný rozdíl je v tom, že zdroj lze číst maximální rychlostí a výstup se ukládá do souboru. Příkaz může vypadat například takto:

```
$ ffmpeg -i vstup.mp3 -f s16le -acodec pcm_s16le -ac 1 -ar 22050 -af
volume=2 vystup.pcm
```

Nejsme však omezeni pouze na PC, konverze se dá provést také přímo na telefonu po nainstalování terminálového emulátoru Termux [23] a instalaci Ffmpeg v Termuxu z jeho repozitáře:

```
$ apt install ffmpeg
```

Nyní je již možné konvertovat soubory stejným způsobem jako na počítači.

5.2.5 Stavový box

Pro snadnější orientaci se vedle tlačítek *Start* a *Stop* nachází stavový box, který trvale zobrazuje, v jakém módu se aplikace nachází. Může zobrazovat následující tři možnosti:

- Stop – činnost aplikace je pozastavena
- ON AIR – probíhá streaming zvuku
- Play – právě probíhá přehrávání souboru.

5.2.6 Stavový řádek

Stavový řádek zobrazuje veškeré důležité informace, jako například informaci o zahájení vysílání nebo ztracení klientské stanice. V tomto případě vypíše i její ID. Též se zde objevují informace o stisknutí tísňového tlačítka. Pří výběru souboru se pro kontrolu zobrazí cesta k vybranému souboru. Novým stiskem tlačítka *Start* se výpis vyčistí.

5.2.7 Oznámení

Během vysílání (ať už zvuku z mikrofonu či nahrávky) je zobrazená permanentní notifikace. Má přitom dva účely. Tím zjevným je informovat uživatele o právě probíhajícím vysílání, zvláště pokud zrovna probíhá na pozadí. Druhým účelem je udržet aplikaci v paměti. U aplikace s permanentní notifikací je menší pravděpodobnost, že ji nativní správce úloh ukončí, kvůli uvolnění chybějících systémových zdrojů.

Další notifikace vznikne při mimořádné situaci, jakou je například ztráta klienta či upozornění na stisknutí tlačítka klientské stanice. Pokud telefon není v tichém režimu, pak se spolu s notifikací přehraje i systémový zvuk upozornění.

5.3 Zdrojový kód

Kód této aplikace se zcela standardně skládá ze tří hlavních částí. Těmi jsou:

- rozložení prvků hlavní obrazovky
- vlastní kód
- manifest

Vzhledem k nulovým zkušenostem vytváření kódu takovéto aplikace jsem si většinou nejdříve vyhledal příklady, kde se podobná problematika řeší a z nich jsem vycházel při řešení konkrétních záležitostí této aplikace.

5.3.1 Použité příklady kódu

Zde jsou zdroje, ze kterých jsem čerpal při vytváření aplikace:

- UDP server pro příchozí komunikaci [24]
- zjištění broadcastové adresy [25]
- zacházení s audio streamem [26]
- změna hlasitosti PCM vzorků [27]
- použité ikony [28]

5.3.2 Popis interních funkcí aplikace

První řádky programu (konkrétně v již zmíněném souboru `MainActivyty.java`) začínají importem knihoven, které jsou z velké části dílem automatického importu, který jsem si zvolil po instalaci Android Studia. Zbytek pochází z kódů, kterými jsem se inspiroval.

Tato sekce slouží především k vyjasnění některých částí kódu, jejichž souhrnný popis by v komentářích zabíral mnoho místa.

Následuje deklarace proměnných, které by někdo mohl chtít při modifikaci aplikace změnit. Sem patří například čísla portů, vzorkovací frekvence zvukového výstupu, výchozí hlasitost a podobně. Následují deklarace globálních proměnných, které musí být přístupné odkudkoli z programu.

Funkce `onCreate()` vytváří nezbytná rozhraní pro obsluhu tlačítek, posuvníku hlasitosti a informační prvky. Deklaruje objekt `NotificationManager` a jím vytvořené jednotlivá oznámení, které se používají dále v aplikaci. Obsahuje vlastní kód pro obsluhu tlačítek a periodicky volá kontrolu jednotlivých stanic.

Obsluha tlačítek většinou znamená (mimo jiné) viditelné zablokování funkcí těch, jejichž využití by v daném kontextu nedávalo smysl. Například po stisknutí tlačítka *Start* dojde k jeho zablokování až do doby, než je vysílání vypnuté.

Kontrola připojených stanic probíhá tak, že v případě přihlášení některé z nich dojde v poli `isConnected[]` ke změně na logickou jedničku u ID příslušné stanice a zároveň v poli `lastSeen[]` uložení aktuálního času (čas, kdy byla stanice naposledy spatřena). Vlastní kontrola potom spočívá v porovnávání těchto uložených hodnot (pouze u stanic s `isConnected[ID] == true`) s aktuálním časem. Pokud je tato hodnota větší než nastavená, spustí se notifikace, dojde k odhlášení stanice (`isConnected[ID] = false;`) a k výpisu záznamu o události do stavového řádku. Opětovným připojením stanice se může celý cyklus opakovat. Po vypnutí vysílání tlačítkem *Stop* se pole `isConnected[]` vynuluje a zamezí se příjmu přihlašovacích paketů. Díky tomuto opatření se při řízeném vypínání setu na telefonu průvodce neobjeví upozornění o vypnutí každé jednotlivé stanice zvlášť.

Vlastní signalizace je poměrně podrobně rozepsána v 3.2.1, takže zde už spíše jen ke zpracování aplikací. Přijaté pakety na port 4445 se kontrolují na velikost a zda-li začínají na příslušnou sekvenci. Pokud nevyhovují oběma požadavkům, jsou zahozeny a dále se jimi program nezabývá. Pokud kritéria splňují, pak se z jejich obsahu získá ID a jsou aktualizovány proměnné `isConnected[]` a `lastSeen[]`. V poslední řadě se zkontroluje, zda-li se jedná o paket pouze ohlašující přítomnost stanice, či jde o upozornění na stisknuté tlačítko.

Funkce `getBroadcast()` slouží ke zjištění broadcastové IP adresy, kterou je důležité znát kvůli vysílání.

Vlastní vysílání probíhá tak, že se do pole `buffer[]` ukládají jednotlivé PCM vzorky získané z objektu `recorder` třídy `AudioRecord` (ze stejnojmenné klihovny). Poté, co se

`buffer` naplní, tak se vzorky přepočítají na novou hlasitost. Tento krok je nezbytný, jelikož získaný zvuk má na většině zařízení velmi nízkou hlasitost. Jen pro představu, v 16-ti bitovém PCM vzorku je v mém případě využíváno pouze spodních 9 bitů (telefon Samsung galaxy S5). Vzhledem k tomu, že jsem nenašel žádné elegantní řešení, které by zahrnovalo ideálně automatické řízení hlasitosti podle vstupní úrovně, násobím každý vzorek zvlášť konstantou. Jedná o 16-ti bitový stream, je tedy nejdříve potřeba přečíst dva byty, které spolu logicky souvisí, vynásobit je určenou hodnotou a opět novou hodnotu rozdělit na dva byty. V posledním kroku je opět ukládám na správné místo v datovém proudu. Takto připravené vzorky už stačí pouze poslat do UDP socketu.

Pokud se průvodce rozhodne poslouchat ve vlastních sluchátkách svůj hlas (použije tlačítko *Playback*), pak se krom vlastního datového vysílání zachycené vzorky přehrají pomocí objektu `audio` třídy `AudioTrack`. Zde není potřeba další zpracování zvuku, použil jsem vzorky určené k vysílání tak jak jsou.

V případě, že se průvodce rozhodne odvysílat zvukový soubor, pak se `recorder` dočasně vyřadí z činnosti a místo toho se data získávají přímo ze souboru s PCM vzorky. Těmi se opět naplní `buffer[]`. Do smyčky bylo také zařazeno čekání. V předchozím případě se o něj staral `recorder`, ale nyní je čtení ze souboru podstatně rychlejší, než potřebných 22050 vzorků za sekundu. Tentokrát není potřeba přepočítávat hlasitost, jelikož vycházíme z předpokladu, že nahrávka byla konvertována do PCM s žádoucím nastavením. Kromě toho, že se takto získaný datový proud odvysílá, je také vynuceno přehrávání stejným způsobem, jako je tomu v předchozím odstavci.

5.4 Návrhy na možná vylepšení

Aplikace se (i přes snahu o maximální funkčnost) v některých ohledech nechová právě tak, jak bych si představoval.

Například při otevření souboru jsem počítal s automatickou konverzí běžně používaných komprimovaných formátů v reálném čase přímo do PCM. Tato možnost se nepodařila implementovat. Původním záměrem bylo také spouštění hlavní aktivity při tapnutí na notifikaci. Při testování tato vlastnost způsobovala nestandardní chování aplikace a proto se v konečné verzi nic nestane.

Zvuk se posílá přes UDP broadcast, což také není ideální řešení. Broadcastové pakety mohou potencionálně zatěžovat i další připojené stanice, byť se s jejich přítomností v konečné konfiguraci nepočítá. Aplikace multicast umí, avšak kvůli tomu, že některé přístroje ho (za použití WiFi – hotspotu) nepodporují, je tato funkcionalita vypnutá.

Oproti plánu se také nepodařilo vyjednávání práv za běhu aplikace u telefonů s Andoridem 6 a vyšším. Doporučený kód [29], pro vyžádání oprávnění, mi ze zatím nezjištěného důvodu nefungoval.

5.5 Aplikace jako mobilní přijímač

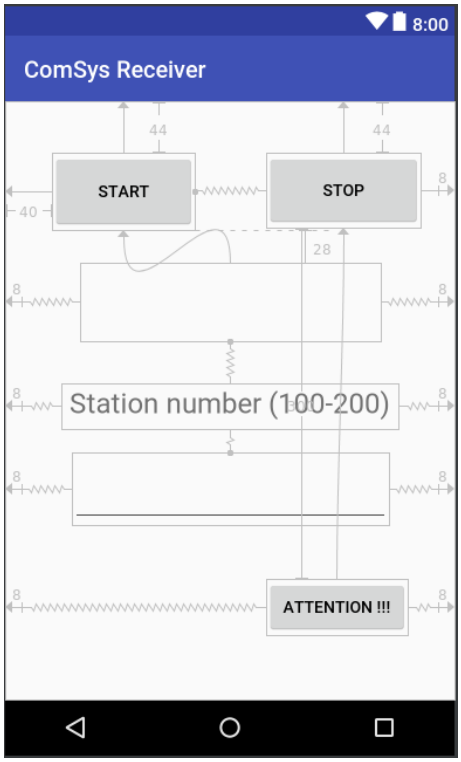
Také jsem měl snahu o vytvoření aplikace, která by fungovala jako mobilní přijímač. Během prvních testů jsem však zjistil, že některé telefony přestanou po vypnutí displeje přijímat broadcastové pakety. Tímto se stává aplikace nepoužitelnou. Na každý takový telefon by bylo nutné posílat vlastní unicastový stream, což by pro vysílající telefon bylo při větším počtu stanic velmi náročné.

Řešením by mohlo být nechat display po celou dobu zapnutý. Nechtěné dotyky pak integrovaným zámkem obrazovky. Problémem je ovšem fakt, že u klasických LCD displejů nelze ve většině případů ztlumit podsvětlení na tak nízkou úroveň, abychom mohli

hovořit o nevýznamné zátěži baterie. Toto řešení by tedy opět bylo vhodné pouze pro část telefonů s OLED displejem. U nich stačí promítnout černou plochu a v takovém případě nespotřebovávají žádnou, nebo téměř žádnou energii.

Z uvedených důvodů jsem vývoj aplikace opustil v momentě, kdy jsem narazil na nemožnost rozumně přijímat broadcastové pakety. Tato aplikace tedy funguje pouze jako přijímač a to jen za předpokladu rozsvíceného displeje či unicastového vysílání. Tlačítko *Start* a *Stop* spustí či ukončí příjem. Ve stavovém boxu se zobrazuje, zda-li se právě naslouchá či nikoli. Možnost zadání ID klientské aplikace je stejně tak jako tlačítko pro pozornost deaktivované. Pokud by se v budoucnu podařilo nějakým elegantním způsobem aplikaci zprovoznit, stačí doplnit funkce pro obsluhu příslušných tlačítek a konečně vlastní tlačítka aktivovat.

Rozvržení ovládacích prvků aplikace je patrné z obrázku 5.2.



Obrázek 5.2. Alikace pro příjem

Kapitola 6
Závěr

V tomto dokumentu jsme si podrobně ukázali, jak funguje navržený HW přijímač i aplikace pro mobilní telefon. Vysvětlili jsme si také, jak probíhá komunikace mezi zařízeními.

Vytvořený prototyp zařízení je funkční a splňuje požadavky uvedené v zadání práce. Během práce na tomto zařízení jsem se naučil používat sockety, zpracovávat data přijatá po síti, práci se zvukovými vzorky a programovat mobilní aplikace v Javě. Také jsem se seznámil se sázecím nástrojem LaTeX a naučil se používat šablonu CTUstyle [30].

Všem, kteří dočetli až sem, děkuji za pozornost. Doufám, že pro vás informace získané v této práci byly užitečné.

Literatura

[1] "Espressif Systems". *"ESP8266EX Datasheet"*. "2018". "https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf" .

[2] "Community of developers". *"ESP8266 Community Forum"*. "2014". "https://www.esp8266.com/" .

[3] "Espressif Systems". *"ESP8266 Developer Zone"*. "2014". "https://bbs.espressif.com/" .

[4] "Ivan Grokhotkov a Michael Molinari". *"ESP8266 Community Forum"*. "2014". "https://github.com/esp8266" .

[5] "Richard Webb". *"ESP8266 Bootloader Modes and GPIO state on Startup"*. "2016". "https://zoetrope.io/tech-blog/esp8266-bootloader-modes-and-gpio-state-startup/" .

[6] "Holtek Semiconductor Inc.". *"HT73XX - Low Power Consumption LDO"*. "2005". "http://www.angeladvance.com/HT73xx.pdf" .

[7] "Espressif Systems". *"SDKs & Demos"*. "2016". "https://www.espressif.com/en/support/download/sdks-demos" .

[8] "Max Filippov". *"Toolchain"*. "2015". "https://github.com/esp8266/esp8266-wiki/wiki/Toolchain" .

[9] "Jeroen Domburg a Jens Hauke". *"I2S MP3 webradio streaming example"*. "2014". "https://github.com/espressif/ESP8266_MP3_DECODER" .

[10] "Paul Sokolovsky". *"esp-open-sdk"*. "2016". "https://cs.wikipedia.org/wiki/SDK" .

[11] "Adam Dunkels a Leon Woestenberg". *"Lightweight IP stack"*. "2004". "http://www.nongnu.org/lwip/2_0_x/index.html" .

[12] "Espressif Systems". *"Gtting started guide"*. "2016". "https://www.espressif.com/en/support/explore/get-started/esp8266/getting-started-guide" .

[13] "Massimo Banzi a David Cuartielles". *"Download the Arduino IDE"*. "2005". "https://www.arduino.cc/en/Main/Software" .

[14] "Jim Lindblom". *"ESP8266 Thing Hookup Guide"*. "2015". "https://learn.sparkfun.com/tutorials/esp8266-thing-hookup-guide/installing-the-esp8266-arduino-addon" .

[15] "Kristijan Gjoshev a Earle F. Philhower". *"i2s.h"*. "2016". "https://github.com/esp8266/Arduino/blob/master/cores/esp8266/i2s.h" .

[16] "Stanislav Mašláň". *"SW emulace sběrnice I2S pro audio DA převodníky"*. "2007". "http://www.elektronika.kvalitne.cz/ATMEL/necoteorie/tutorial/AudioTest/I2Semulation/I2S_emulation.html" .

[17] "Fabrice Bellard". *"FFmpeg"*. "2007".
"https://www.ffmpeg.org/" .

[18] "Tomasz Ostrowski". *"ESP8266 VoIP/RTP pager"*. "2016".
"http://tomeko.net/projects/rtp_pager/" .

[19] "Erik H. Bakke". *"Reducing WiFi power consumption on ESP8266*.
"https://www.bakke.online/index.php/2017/05/21/reducing-wifi-power-consumption-on-esp8266-part-1/" .

[20] "Espressif Systems". *"ESP8266 Power Consumption"*. "2015".
"https://bbs.espressif.com/viewtopic.php?t=133" .

[21] Karel Zadražil. *Zařízení pro charakterizaci zdrojů stejnosměrného napětí*. 2016.
"https://dspace.cvut.cz/handle/10467/64689" .

[22] "Google Inc.". *"Android Studio"*. "2013".
"https://developer.android.com/studio/" .

[23] "Fredrik Fornwall". *"Termux"*. "2015".
"https://termux.com/" .

[24] "Android Erik (pseudonym)". *"Android Datagram/UDP Server example"*. "2016".
"http://android-er.blogspot.cz/2016/06/android-datagramudp-server-example.html" .

[25] "Cristian (pseudonym)". *"Android Broadcast Address"*. "2010".
"https://stackoverflow.com/questions/2993874/android-broadcast-address" .

[26] "Hugues Verlin a chuckliddell0 (pseudonym)". *"Stream Live Android Audio to Server"*. "2013".
"https://stackoverflow.com/questions/15349987/stream-live-android-audio-to-server/19756061" .

[27] "Hartmut Pfitzinger". *"Increase volume output of recorded audio"*. "2014".
"https://stackoverflow.com/questions/26088427/increase-volume-output-of-recorded-audio" .

[28] "Creativetail". *"Free Icons - Collection Of Simple Icons*.
"www.creativetail.com" .

[29] "Android.com". *"Permissions Overview"*. "2018".
"https://developer.android.com/guide/topics/permissions/overview" .

[30] "Petr Olšák". *"CTUstyle šablona pro sazbu studentských závěrečných prací na ČVUT"*. "2013".
"http://petr.olsak.net/ctustyle.html" .

Příloha A
Seznam zkratek

AP	Přístupový bod (Acces Point)
API	Rozhraní pro programování aplikací (Application Programming Interface)
DAC	Převodní digitálního signálu na analogový (Digital to Analog Converter)
DHCP	Protokolu z rodiny TCP/IP pro automatickou konfiguraci počítačů (Dynamic Host Configuration Protocol)
ESP	Čip ESP8266
FW	Program mikroprocesoru (Firmware)
GPIO	Vstup/výstup pro univerzální použití (General-Purpose Input / Output)
HTTP	Protokol určený pro výměnu hypertextových dokumentů (Hypertext Transfer Protocol)
HW	Hardware
I/O	Vstupně výstupní piny (Input / Output)
ID	Identifikátor (Identifier)
IDE	Vývojové prostředí (Integrated Development Environment)
IP	Skupina síťových protokolů (Internet Protocol)
I2S	Digitální audio sběrnice pro komunikaci integrovaných obvodů (Inter-IC Sound)
LCD	Displej z tekutých krystalů (Liquid Crystal Display)
LED	Světelná dioda (Light Emitting Diode)
LwIP	Odlehčená implementace TCP/IP stacku (lightweight IP)
OLED	Displej z organických LED diod (Organic Light Emitting Diode)
OS	Operační systém (Operating System)
OTA	Bezdrátová aktualizace firmware (Over-The-Air update)
PCB	Deska plošných spojů (Printed Circuit Board)
PCM	Pulzně-kódová modulace (Pulse-Code Modulation)
PWM	Pulzně šířková modulace (Pulse Width Modulation)
RAM	Paměť s náhdoný přístupem (Random Access Memory)
RTC	Hodiny reálného času (Real Time Clock)
RX	Příjem (Receive)
SDK	Sada vývojových nástrojů (Software Development Kit)
SPI	Sériové periferní rozhraní (Serial Peripheral Interface)
UART	Univerzální asynchronní sériové rozhraní (Universal Asynchronous Receiver and Transmitter)
URI	Jednotný identifikátor zdroje (Uniform Resource Identifier)
URL	Řetězec pro identifikaci dokumentu v internetu (Uniform Resource Locator)
USB	Univerzální sériová sběrnice (Universal Serial Bus)

Příloha B
Seznam souborů

Nedílnou součástí této práce je soubor `priloha.zip` obsahující vytvořené aplikace a zdrojové kódy. Níže v tabulce naleznete jejich seznam s podrobnějším popisem.

Název souboru	Vysvětlivka
`ComSys.apk`	Průvodcovská aplikace pro vysílání
`ComSys.zip`	Zdrojové kódy průvodcovské aplikace
`ComSysReciever.apk`	Klientská aplikace pro příjem
`ComSysReceiver.zip`	Zdrojové kódy klientské aplikace
`ComSys.ino`	Zdrojový kód HW přijímače
`Logger.ino`	Zdrojový kód dataloggeru použitého v 4.2.2

Tabulka B.1. Tabulka přiložených souborů