České Vysoké Učení Technické v Praze
Fakulta elektrotechnická
Katedra mikroelektroniky

Diplomová práce

Zpracování signálu mikrokontrolérem s využitím metod strojového učení

Bc. Michael Funderák
Elektronika a komunikace

Vedoucí
doc. Ing. Stanislav Vítek, Ph.D.

Praha 2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Funderák** Jméno: **Michael** Osobní číslo: **465992**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra mikroelektroniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Elektronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Zpracování signálu mikrokontrolérem s využitím metod strojového učení

Název diplomové práce anglicky:

Signal Processing on Microcontroller Using Machine Learning Methods

Pokyny pro vypracování:

1) Proveďte rešerši metod zpracování signálu s využitím metod strojového učení.
2) Prostudujte možnosti implementace metod z bodu (1) na mikrokontrolérech třídy STM32F4x.
3) Navrhněte a implementujte externí periferii mikrokotroléru pro směrové snímání zvuku.
4) Implementujte algoritmus pro identifikaci a klasifikaci zdroje zvuku.
5) Diskutujte výsledky svého řešení.

Seznam doporučené literatury:

[1] WARDEN, Pete; SITUNAYAKE, Daniel. Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers. O'Reilly Media, 2019.
[2] DUTTA, Lachit; BHARALI, Swapna. TinyML Meets IoT: A Comprehensive Survey. Internet of Things, 2021, 16: 100461.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Stanislav Vítek, Ph.D. katedra radioelektroniky FEL

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

Datum zadání diplomové práce: **09.02.2022** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2023**

doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Pavel Hazdra, CSc. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. 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.

. _____
Datum převzetí zadání

_____
Podpis studenta

Prohlášení

Prohlašuji, že jsem zadanou diplomovou práci zpracoval sám s přispěním vedoucího práce doc. Ing. Stanislava Vítka, Ph.D. a používal jsem pouze literaturu v práci uvedenou. Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé diplomové práce nebo její části se souhlasem katedry.

V Praze 25.5.2023

.....
Bc. Michael Funderák

Poděkování

Rád bych poděkoval především doc. Ing. Stanislavu Vítkovi Ph.D. za podporu, vedení a trpělivost v průběhu celého projektu. Dále bych chtěl poděkovat své rodině a přátelům za podporu po dobu celého studia.

Abstrakt

Tato práce se zabývá klasifikací a lokalizací zdroje zvuku pomocí mikrokontroléru STM32F411RE. Zvuk je zaznamenán čtyřmi mikrofony SPH0645lm4h.

Lokalizace zdroje zvuku používá zobecněnou křížovou korelaci s fázovou transformací. Digitální zpracování signálu je provedeno metodou Melových frekvenčních kepstrálních koeficientů. Klasifikaci zvuku spravuje natrénovaný model konvoluční neuronové sítě, který je implementován na mikrokontrolér. V rámci práce je navržena deska plošných spojů s mikrokontrolérem, mikrofony a osmi LED, které signalizují směr zdroje zvuku sirény.

Klíčová slova: Mikrokontrolér, lokalizace zvuku, MFCC, CNN, GCC, PHAT

Abstract

The main goal of the diploma thesis is to classify and localize the sound source using STM32F411RE microcontroller. The sound is recorded using the four microphones of the SPH0645lm4h.

Sound source localization is using generalized cross-correlation with phase transformation. For digital signal processing the method Mel frequency cepstral coefficients is used. Sound classification is managed by trained neural network model, which is implemented on a microcontroller. A printed circuit board is designed, featuring a microcontroller, microphones and eight LEDs that indicate the direction of the sound source of the siren.

Key words: Microcontroller, sound localization, MFCC, CNN, GCC, PHAT

Obsah

Seznam obrázků

Seznam tabulek

Seznam symbolů a zkratek

Seznam zkratek

MFCC	Melovy frekvenční kepstrální koeficienty
DFT	Diskrétní Fourierova transformace
DCT	Diskrétní Kosinova transformace
GCC	Zobecněná křížová korelace
PHAT	Fázová transformace
MEMS	Mikro elektromechanický systém
ADC	Analagově číslicový převodník
MSB	Nejvíce signifikantní bit
SWD	Sériový debugger
CPU	Centrální procesorová jednotka
RC	Rezistor, kapacitor
PLL	Fázový závěs
ID	Identifikační číslo
DSP	Digitální zpracování signálu
GUI	Grafické uživatelské rozhraní
SRAM	Statická paměť s náhodným přístupem
CMSIS	Standard pro společné rozhraní softwaru mikrokontrolérů
DMA	Přímý přístup do paměti
HAL	Hardwarová abstrakční vrstva
GPIO	Univerzální vstup a výstup
I^2S	Zvukové rozhraní pro integrované obvody
WS	Synchronizační signál
CK	Hodinový signál
SD	Datový signál
SPI	Sériové periferní rozhraní
USART	Univerzální synchronní a asynchronní přijímač a vysílač
TX	Vysílač
RX	Přijímač
LED	Dioda vyzařující světlo
USB	Univerzální sériová sběrnice
LDO	Lineární regulátor s nízkým
ESC	Klasifikátor okolních zvuků
MCU	Mikrokontrolér
FW	Firmware
CNN	Konvoluční neuronová síť

Kapitola 1
Úvod

V posledních letech se digitální zpracování signálu a strojové učení staly klíčovými technologiemi v oblasti elektroniky. Současně rostou požadavky na implementaci digitálního zpracování signálu a strojového učení do vestavěných systému na mikrokontrolérech. Jejich výhodou je především nízká spotřeba energie, malý formát a cenová dostupnost, což z nich dělá ideální kandidáty pro řadu aplikací v oblasti Internetu věcí, dále jen IoT. Mikrokontroléry hrají klíčovou roli v ekosystému IoT od sbírání dat, přes řízení aktuátoru až po přenos dat. Koncový článek IoT sítě se skládá právě z mikrokontroléru a senzoru nebo aktuátoru. Nasbíraná data jsou přeposílána na cloudové uložiště, které zastupuje roli centrálního mozku. Na uložišti jsou zpracována data a pomocí strojového učení jsou učiněna inteligentní rozhodnutí.

Zpracování dat přímo na koncovém uzlu mikrokontrolérem se zdá být efektivnější než přeposílání na cloud. I když má uložiště enormní výpočetní sílu, zpracování a vyhodnocení nově příchozích dat zabere mnoho výpočetního času a energie. Následkem je zpomalení celé IoT sítě.

V posledním desetiletí byl zaznamenán úspěch v oboru hlubokého učení, neuronových sítí a posilovaného učení. Nicméně jejich použití bylo limitováno na grafické karty, datová centra a superpočítače. Hlavním faktorem, který dříve bránil používání algoritmů strojového učení na mikrokontrolérech, bylo převedení velkého množství dat do malého pamětního prostoru.

TinyML je nový směr v oblasti strojového učení. Zaměřuje se na implementaci a provádění výpočtů strojového učení přímo na malých a energeticky úsporných MCU. Cílem je provádět inteligentní a sofistikované algoritmy na okraji IoT sítě. Frekvence přenášení dat se zmenší a zrychlí se celý systém. TinyML poskytuje kompaktní, optimalizované a energeticky efektivní modely strojového učení, které mohou být spuštěny přímo na zařízeních s omezenými zdroji. Využity jsou pokročilé techniky komprese modelů, kvantizace vah, optimalizace inferenčních algoritmů a hardwarová akcelerace.

Použití TinyML má širokou škálu aplikací v oblasti IoT a vestavěných systémů. Může se jednat o monitorovací senzory, zařízení pro analýzu zvuku a obrazu, systémy pro rozpoznávání gest nebo průmyslovou automatizaci. TinyML otevírá nové možnosti vytváření inteligentních zařízení, která mohou provádět analýzu a rozhodnutí na základě svého okolí v reálném čase [1, 2].

Vozidla integrované záchranné služby na sebe upozorňují sirénou, blikajícím modrým světlem a specifickým barevným označením. Brzkou lokalizací sirény mohou účastníci silničního provozu bezpečně a rychle uvolnit místo záchrannému vozidlu. Chybným určením směru můžou řidiči a chodci zareagovat pomalu a dopouštět se chyb. To může mít za následek zpomalení vozidla záchranné služby a vytvoření potenciálního ne-

bezpečí pro všechny zúčastněné. Odhlučnění automobilu, hrající rádio a hluk z vnějšího urbanistického prostředí snižují schopnost člověka správně určit směr zdroje sirény. S narůstajícím počtem automobilů na silnici je důležité vytvořit spolehlivý systém, který řidiče upozorní na přibližující se vozidlo integrované záchranné služby [3].

1.1 Cíle práce

Cílem této diplomové práce je implementovat neuronovou síť, která bude schopna rozpoznávat zvuk sirény pomocí MCU a určovat, z jakého směru zvuk přichází.
Konkrétně se jedná o následující cíle:

1. Implementace neuronové sítě: Navrhnout a implementovat neuronovou síť vhodnou pro rozpoznává zvuku sirény.
2. Směrové snímání zvuku: Navrhnout a implementovat metodu pro směrové snímání zvuku založenou na použití mikrofonů. Cílem je získat informaci o směru, ze kterého zvuk sirény přichází.
3. Integrace na mikrokontrolér: Implementovat navrženou neuronovou síť a metodu směrového snímání zvuku na mikrokontrolér. Mikrokontrolér umožní výpočet a rozpoznávání zvuku přímo na zařízení.
4. Validace výsledků: Provést experimenty a validaci implementovaného systému. Vyhodnotit přesnost rozpoznávání zvuku sirény a správnost určení směru zvuku na základě naměřených dat.

Kapitola 2
Teoretický základ

2.1 Strojové učení

Strojové učení je obor umělé inteligence, který umožňuje počítačům učit se, dělat predikce nebo rozhodnutí bez explicitního programování. Obor se opírá o statistické techniky a výpočetní algoritmy k rozpoznání vzorců a vztahů v datech. Strojové učení se dělí na učení s učitelem a bez učitele.

Učení s učitelem trénuje model na základě manuálně označených dat. Pro každý vstupní příklad je znám jeho výstup. Učení s učitelem se využívá pro vytvoření regresních nebo klasifikačních modelů.

Učení bez učitele se zaměřuje na analýzu neoznačených dat. Hlavním cílem učení bez učitele je objevení skrytých vzorců, shlukování (*clustering*) podobných dat nebo redukce dimenzionality [4].

2.1.1 Klasifikace

Klasifikace je v oblasti strojového učení proces, který přiřazuje objekty do předem definovaných tříd na základě jejich vlastností. Cílem klasifikace je vytvořit model, který se naučí rozpoznávat vzorce mezi vstupními daty a jejich příslušnými třídami. Naučený model pak může být použit ke klasifikaci nových, dosud neoznačených vstupů. Velmi často využívaný případ je single-label klasifikace, kde každému vstupu z trénovací množiny je přiřazena právě jedna třída.

Model lze zhodnotit různými metrikami. Nejčastěji se používá přesnost (*accuracy*), která udává poměr správně klasifikovaných příkladů ke všem příkladům. Hodnota vyjadřuje, jak často model správně predikuje třídy, viz rovnici 2.1. Přesnost můžeme vyhodnotit i pro každou třídu zvlášť a tím odhalit, zda je model přesný v klasifikaci všech tříd.

$$Accuracy = \frac{TP + TN}{P + N} \tag{2.1}$$

- TP je počet správně určených pozitivních výsledků ($True\ Positive$)
- TN je počet správně určených negativních výsledků ($True\ Negative$)
- P je počet všech pozitivních výsledků ($Positive$)
- N je počet všech negativních výsledků ($Negative$)

Preciznost (*precision*) je metrika, která měří podíl správných pozitivních klasifikací vůči všem klasifikacím, viz rovnici 2.2. Senzitivita (*sensitivity*) je metrika, která měří podíl správných pozitivních klasifikací vůči všem skutečným pozitivním výsledkům, viz rovnici 2.3. Obě metriky jsou užitečné při určení falešně pozitivních a falešně negativních výsledků.

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

- FP je počet falešně pozitivních výsledků ($False\ Positive$)

$$Sensitivity = \frac{TP}{P} \tag{2.3}$$

Matice záměn ($confusion\ matrix$) zobrazuje počty správně a nesprávně klasifikovaných příkladů pro každou třídu. Matice vizualizuje oblasti, ve kterých model dělá chyby [5].

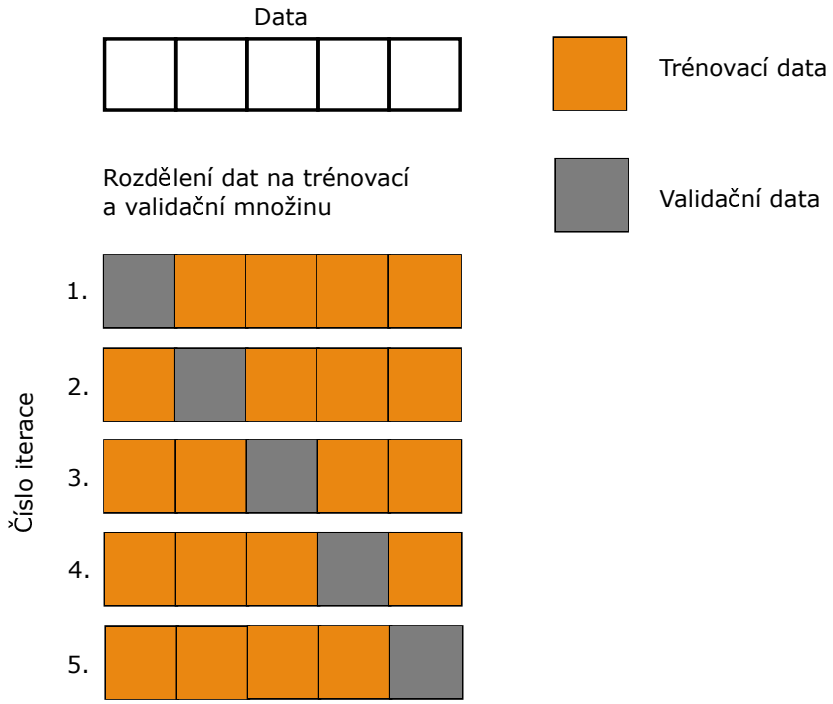
2.1.2 Trénování modelu

Cílem trénovacího procesu je vytvořit model, který dokáže správně klasifikovat neznámá vstupní data. Před začátkem tréninku je proto zapotřebí data připravit a předzpracovat. Na základě povahy problému a dostupných dat je vybrán vhodný model a použitý algoritmus. Model je před trénováním inicializován náhodnými váhami a parametry. Tyto počáteční hodnoty slouží jako výchozí bod pro optimalizační proces. Model je trénován iterativně pomocí trénovacích dat. Během každé iterace model provádí predikce na těchto datech a rozdíl mezi predikovanými a skutečnými výstupy je měřen pomocí ztrátové funkce. Parametry modelu jsou během trénování upravovány za účelem minimalizace ztrátové funkce. Obvykle je úprava prováděna pomocí optimalizačních algoritmů. Natrénovaný model je vyhodnocen na samostatné validační sadě, které model nezná. Při neuspokojivých výsledcích je potřeba měnit hyperparametry, například architekturu modelu, regulační parametry nebo rychlost učení (*learning rate*).

Důležitými pojmy v trénování je přeučení ($overfit$) a podučení ($underfit$) neuronové sítě. Model je přeučený ve chvíli, kdy je příliš přizpůsobený trénovacím datům a nedokáže dobře generalizovat nová data. Přeučení se projevuje vysokým rozdílem přesností trénovacích a validačních dat. Podučení je situace, kdy je model příliš jednoduchý a nedokáže podchytit složitost problému. Výsledkem je nízká přesnost na trénovacích i validačních datech [6].

2.1.3 Rozdělení datasetu

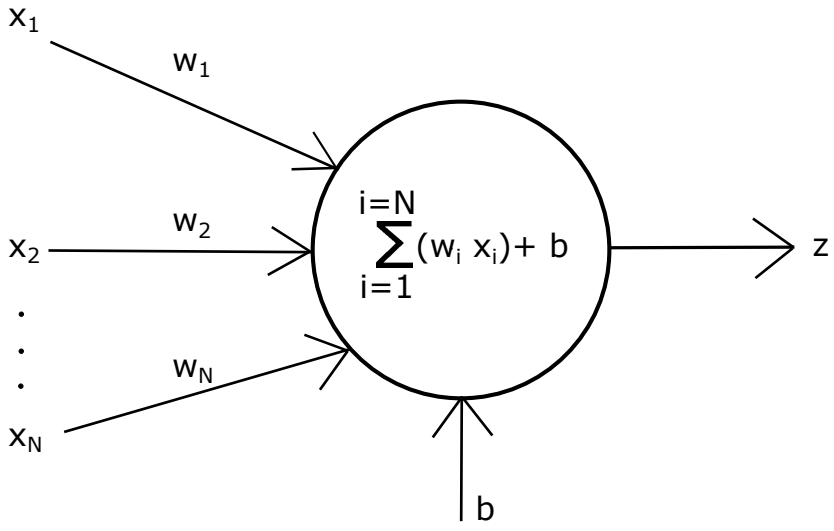
V oblasti strojového učení se běžně používá rozdělení datasetu na trénovací a validační sadu. Metoda holdout rozdělí náhodně dataset na validační a trénovací část. Typické rozdělení je 80 % trénovacích dat a 20 % validačních. Při křížové validaci je dataset rozdělen na K skupin (složek). Pro každou skupinu se model natrénuje na všech ostatních skupinách a validace je vyhodnocena na dané skupině. Výsledkem křížové validace je průměr přesností ze všech skupin a směrodatná odchylka. Nejčastěji se můžeme setkat s desetinásobnou nebo pětinásobnou křížovou validací [7].



Obrázek 2.1: Ukázka pětinásobné křížové validace

2.2 Neuronové sítě

Neuronové sítě představují široce využívaný přístup v oblasti strojového učení a umělé inteligence. Tyto sítě jsou inspirovány biologickým fungováním mozku. Umělý neuron je základním stavebním kamenem neuronových sítí. Přijímá vstupy x_1 až x_N z jiných neuronů nebo vnějšího prostředí a na základě vstupů generuje výstupní signál z. Výstup neuronu je kombinací lineárního zpracování vstupních dat pomocí váhových parametrů w_1 až w_N a nelineární aktivační funkce. Nejjednodušším modelem neuronové sítě je perceptron. Perceptron je sestaven pouze z jednoho neuronu, který má funkci binárního lineárního klasifikátoru. Jeho funkce je popsána obrázkem 2.2 a rovnicí 2.4 [8].



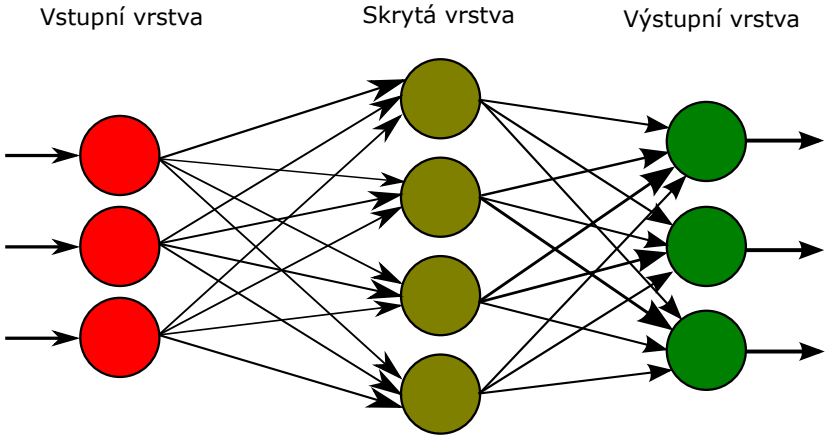
Obrázek 2.2: Perceptron složený z umělého neuronu

$$f(\boldsymbol{x}) = \begin{cases} 1, & \text{pro } \sum_{i=1}^{i=N}(w_i \cdot x_i) - b \geq 0 \\ 0, & \text{jindy} \end{cases} \tag{2.4}$$

- $\boldsymbol{x}$ je vektor vstupních dat neuronu s elementy $(x_1, x_2, ...x_N)$
- $\boldsymbol{w}$ je váhový vektor s elementy $(w_1, w_2, ...w_N)$

2.3 Vícevrstvá perceptronová síť

Vícevrstvá perceptronová síť je nejčastěji používaná architektura neuronových sítí. Struktura je založena na konceptu umělých neuronů, viz obrázek 2.2, které jsou mezi sebou propojeny a pracují společně na zpracování informací. Vícevrstvá perceptronová síť se skládá z minimálně tří vrstev. Vstupní vrstva přijímá vnější data. Počet neuronů ve vstupní vrstvě odpovídá dimenzi vstupních dat. Prostřední vrstvou je skrytá vrstva. Jejím hlavním úkolem je transformovat vstupní data a vytvořit různé abstraktní reprezentace. Každý neuron ve skryté vrstvě přijímá vstup od všech neuronů z předchozí vrstvy a jeho výstup je předán všem neuronům v následující vrstvě. Výstupní vrstva představuje výstup celé sítě. Každý neuron reprezentuje jednu klasifikovanou třídu. Výstupní hodnoty lze interpretovat jako pravděpodobnostní odhady jednotlivých tříd. Vícevrstvá perceptronová síť se učí pomocí algoritmu zpětného šíření (*backpropagation*), který aktualizuje váhy jednotlivých neuronů na základě rozdílu mezi skutečným a předpovězeným výstupem sítě.



Obrázek 2.3: Vícevrstvá perceptronová síť [9]

2.4 Aktivační funkce

Aktivační funkce určují, jakým způsobem neuron reaguje na vstup a jak bude generován výstup. Důležitou vlastností aktivačních funkcí je nelinearita, díky které jsou neuronové sítě schopny aproximovat libovolně složité funkce a řešit komplexní úlohy.

Funkce ReLU (Rectified Linear Unit) je jednou z nejpoužívanějších aktivačních funkcí pro skryté vrstvy v neuronových sítích. Rovnice 2.5 a obrázek 2.4 popisují chování funkce ReLU.

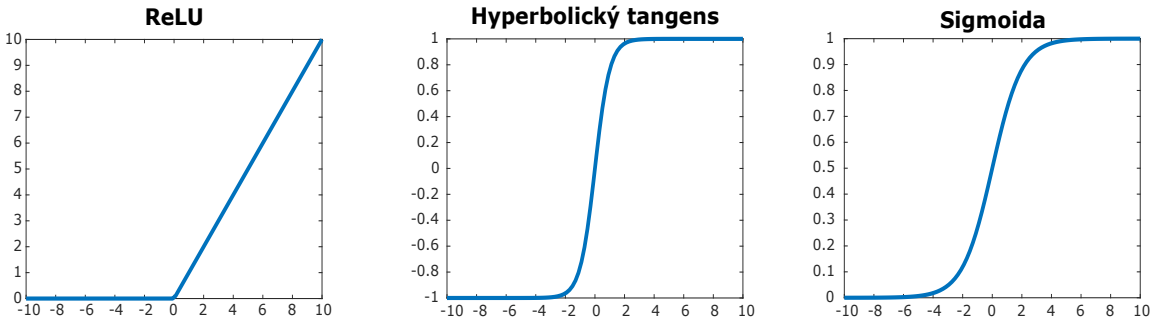
$$f(x) = \begin{cases} x, & \text{pro } x \geq 0 \\ 0 & \text{jindy} \end{cases} \tag{2.5}$$

Hyperbolický tangens je aktivační funkce, která převede vstupní hodnoty na rozmezí (-1,1). Funkce je velice užívaná ve skrytých vrstvách neuronových sítí. Rovnice 2.6 a obrázek 2.4 definují hyperbolický tangens.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.6}$$

Sigmoidní aktivační funkce (také nazývána jako logistická) převede vstupní hodnoty na rozmezí (0,1). Funkce se používá v skrytých i výstupních vrstvách. Sigmoidní funkce je popsána rovnicí 2.7 a obrázkem 2.4.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$



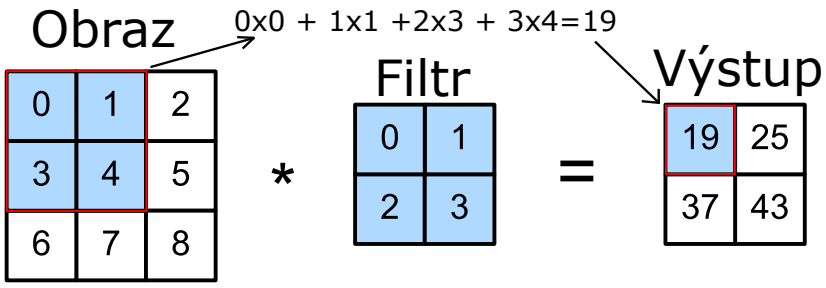
Obrázek 2.4: Grafické zobrazení aktivačních funkcí

Funkce softmax je speciální aktivační funkce využívaná zejména při klasifikacích s více třídami. Softmax převede vstupní hodnoty na pravděpodobnostní distribuci, kde součet všech pravděpodobností je roven jedné. Tím interpretuje výstupy jako pravděpodobnostní odhady různých tříd. Rovnice 2.8 definuje funkci softmax.

$$f(x_i) = \left\{ \frac{e^{x_i}}{\sum_{j=1}^{j=N} e^{x_j}}, \quad \text{kde } \boldsymbol{x} \in \{x_1, x_2 ... x_N\} \right. \tag{2.8}$$

2.5 Konvoluční vrstva

Konvoluce je matematická operace, která je často používána při zpracování obrazů. V kontextu s konvoluční vrstvou a tedy i konvoluční neuronovou sítí je tato matematická operace základní blokem pro získání informace o vlastnosti vstupních dat. Během konvoluce se používá filtr (někdy zvaný jako jádro). Jedná se o malou matici (typicky 3x3 nebo 5x5), která obsahuje váhovací parametry. Filtr se posouvá po vstupních datech daným krokem. Na každé pozici jsou vstupní data násobena váhou příslušného filtru. Součiny jsou sečteny a výsledek součet uložen ,viz obrázek 2.5 [6].

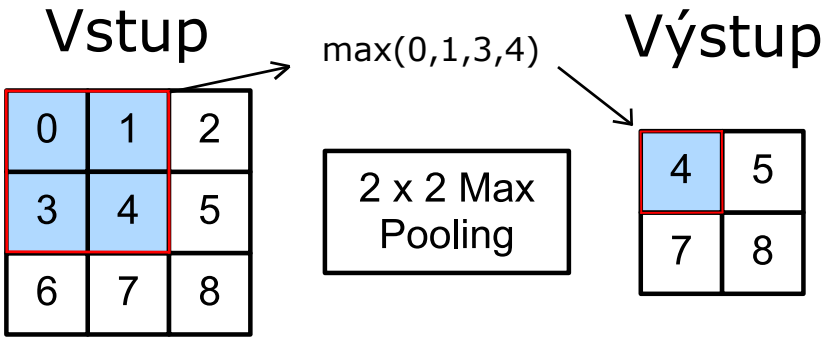


Obrázek 2.5: Filtrování obrazu pomocí konvoluce [6]

2.6 Redukce dimenze v konvolučních neuronových sítích

Snížení dimenzionality (*pooling*) se v konvolučních sítích užívá k redukci parametrů a výpočetní náročnosti sítě. Výstupy sítí jsou více robustní proti malým změnám v obrazu. Snížení dimenze vybírá dominantní vzorce obrazu. Tím jsou zachyceny pouze nejdůležitější informace. Výsledkem je lepší schopnost sítě generalizovat vstupní data a odolávat případnému přeučení.

Vrstva Maximum pooling snižuje dimenzi pomocí jádra, které se pohybuje po vstupu stejným způsobem jako konvoluční jádro. Pro každou pozici vybere jádro maximální hodnotu z oblasti, kterou na vstupu zabírá, viz obrázek 2.6.



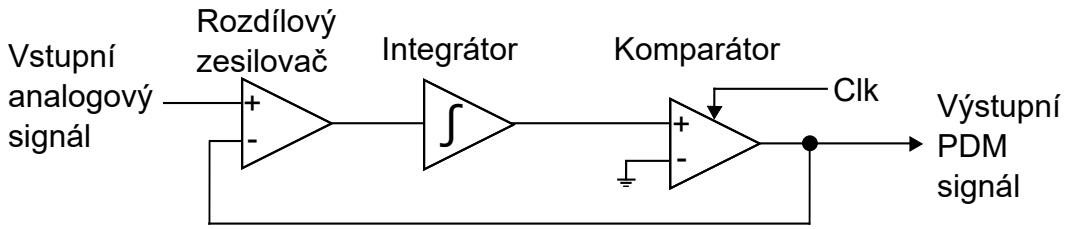
Obrázek 2.6: Snížení dimenze pomocí vrstvy Maximum pooling [6]

2.7 Zpracování zvuku

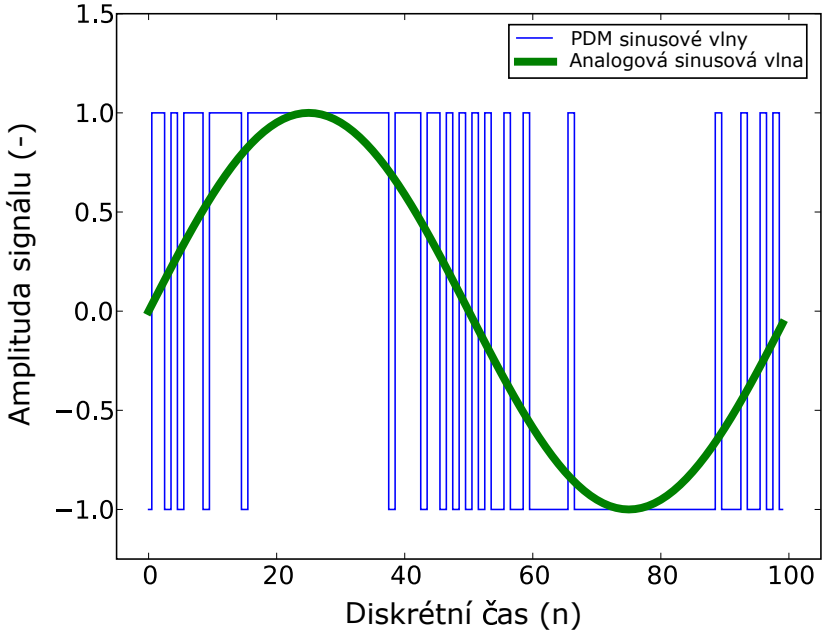
2.7.1 Záznam zvuku a jeho digitalizace

Zvuk je nahráván pomocí mikrofonu. Jedná se o senzor, který převádí změnu akustického tlaku na elektrické napětí. Existuje mnoho typů mikrofonů, nicméně v digitálním zpracování zvuku se nejčastěji používají mikrofony s digitálním výstupem.

Výstupem digitálních mikrofonů je ve většině případů signál PDM (*Pulse Density Modulation*). Při procesu PDM je analogový signál nejprve vzorkován pomocí delta-sigma převodníku, viz obrázek 2.7. Každý vzorek je reprezentován jedním jednobitový číslem, které vyjadřuje přítomnost signálu v daném okamžiku. Vyšší hustota pulzů značí vyšší amplitudu signálu viz obrázek 2.8 [10].

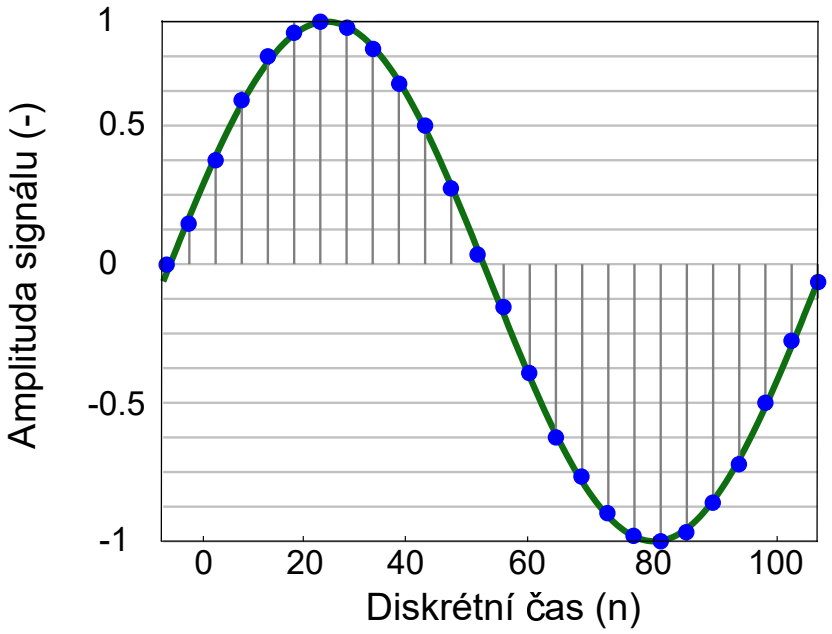


Obrázek 2.7: Schéma delta-sigma převodníku



Obrázek 2.8: Příklad jedné periody sinusového signálu a jeho převodu do PDM [11]

Signál PCM (*Pulse Code Modulation*) je digitální reprezentace zvuku v bytech. Tento formát využívají digitální audio formáty, například WAV, MP3. Formát PCM získáme snížením vzorkovací frekvence PDM signálu pomocí decimačního filtru. Na orbázku 2.9 je zobrazen navzorkovaný sinusový signál ve formátu PCM.



Obrázek 2.9: Příklad jedné periody navzorkovaného sinusového signálu pomocí PCM [12]

2.7.2 Melovy frekvenční kepstrální koeficienty

Melovy frekvenční kepstrální koeficienty (dále jen MFCC) jsou reprezentací krátkodobého výkonového spektra zvuku, založeného na diskrétní kosinové transformaci logaritmu výkonového spektra, které se nachází na nelogaritmické Melově frekvenční škále.

Melova frekvenční škála přibližně modeluje citlivost lidského ucha. Pod 1 kHz je přibližné lineární, nad tímto prahem má logaritmický tvar. Analýza v Melově škále je široce využívaná v moderních algoritmech pro rozpoznávání řeči. Rovnice 2.9 reprezentuje její aproximaci.

$$B(f) = 1125 \cdot ln(1 + \frac{f}{700}) \tag{2.9}$$

K získání frekvenčního spektra je potřeba převést digitalizovaný zvuk do frekvenční domény pomocí diskrétní Fourierovy transformace, dále jen DFT. Rovnice 2.10 je aplikována na jedno analyzovaný rámec (okénko).

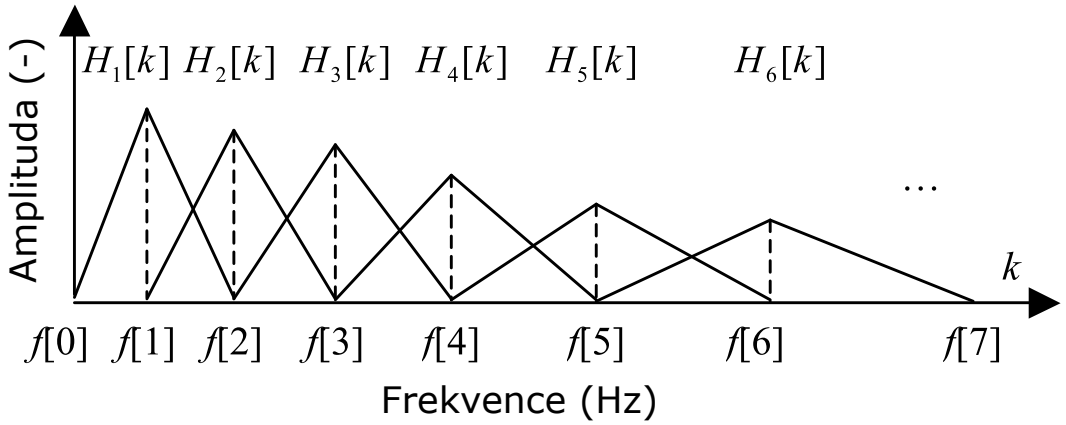
$$X_a[k] = \sum_{n=0}^{N-1}(x_a[n] \cdot e^{\frac{-j \cdot 2\pi \cdot n \cdot k}{N}}), \text{ kde } 0 \leq k < N \tag{2.10}$$

- a je index analyzovaného rámce
- $x[n]$ je zaznamenaný zvuk v časové doméně
- k je index binu DFT
- K je celkový počet binů DFT, délka DFT

Signál ve frekvenční doméně je potřeba převést na výkonové spektrum, viz rovnici 2.11.

$$P_a[k] = \frac{1}{N} \cdot |X_a[k]|^2 \tag{2.11}$$

Dalším krokem je spočítání M triangulačních filtrů ($m = 1, 2, \cdots, M$), které jsou součástí filtrační banky. Obrázek 2.10 reprezentuje vymodelované filtry pomocí rovnice 2.12.



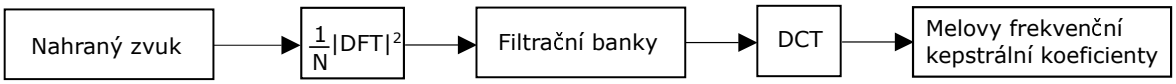
Obrázek 2.10: Triangulární filtrační banky dopočítané pomocí rovnice 2.12 [13]

$$H_m[k] = \begin{cases} 0 & \text{pro } k < f[m-1] \\ \dfrac{2 \cdot (k - f[m-1])}{(f[m+1] - f[m-1]) \cdot (f[m] - f[m-1])} & \text{pro } f[m-1] \leq k \leq f[m] \\ \dfrac{2 \cdot (f[m+1] - k)}{(f[m+1] - f[m-1]) \cdot (f[m+1] - f[m])} & \text{pro } f[m] \leq k \leq f[m+1] \\ 0 & \text{pro } k > f[m+1] \end{cases} \tag{2.12}$$

- m je index triangulačního filtru
- $f[m-1]$ je frekvence, na které má triangulační filtr m nulovou amplitudu. Od této frekvence do frekvence $f[m]$ amplituda filtru lineárně roste.
- $f[m]$ je frekvence, na které má triangulační filtr m nejvyšší amplitudu. Od této frekvence do frekvence $f[m+1]$ amplituda filtru lineárně klesá
- $f[m+1]$ je frekvence, od které triangulační filtr m má nulovou amplitudu a dále již nebude propouštět signál

Výkonové spektrum je filtrováno jednotlivými bankami. Všech M výsledků je zlogaritmováno. Posledním krokem je provedení diskrétní kosinovy transformace, dále jen DCT. Její rovnice je znázorněna v 2.13. Celý postup je výpočtu MFCC je zaznamenán na obrázku 2.11 [13].

$$C[k] = \sum_{n=0}^{N-1} (x[n] \cdot cos(\pi \cdot k \cdot \frac{(n + \frac{1}{2})}{N}) \text{ pro } 0 \leq k < N \tag{2.13}$$



Obrázek 2.11: Blokové schéma postupu pro výpočet MFCC

2.8 Lokalizace zvuku

Zvukový signál šířící se ze vzdáleného zdroje zvuku je monitorován dvěma mikrofony. Rovnice 2.14 a 2.15 popisují matematický model takové situace.

$$x_1(t) = s_1(t) + n_1(t) \tag{2.14}$$

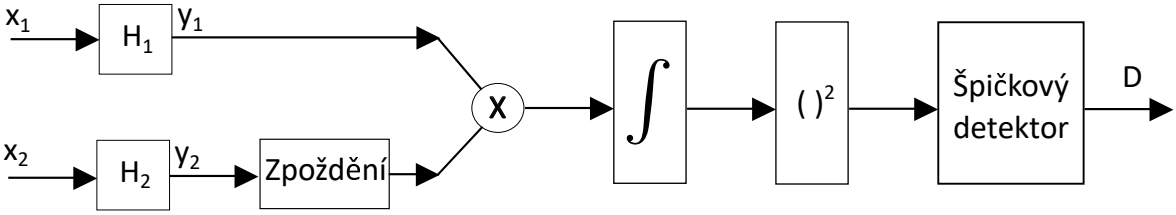
$$x_2(t) = \alpha s_1(t + D) + n_2(t) \tag{2.15}$$

- $s_1(t)$ je signál vyslaný zdrojem zvuku
- $n_1(t)$, $n_2(t)$ jsou šumy
- D je zpoždění signálu $s_1(t)$ k mikrofonu, kterému je přiřazen výstupní signál $x_2(t)$

Pro odhad směru šíření zdroje zvuku vzhledem k mikrofonům je potřeba odhadnout zpoždění přijatého signálu $x_1(t)$ vůči přijatému signálu $x_2(t)$. Jedna z metod určujících časové zpoždění D dvou signálů je křížová korelace, kterou lze získat pomocí rovnice 2.16.

$$R_{x_1x_2}(\tau) = \int_{-\infty}^{\infty} x_1(t) \cdot x_2(t - \tau)dt \tag{2.16}$$

Pro zlepšení výsledků je žádoucí před začátkem integrace 2.16 signály $x_1(t)$, $x_2(t)$ filtrovat skrze filtry H_1, H_2, viz obrázek 2.12. Výsledkem jsou signály $y_1(t)$ a $y_2(t)$, které mezi sebou vynásobíme, zintegrujeme a umocníme na druhou. Pomocí špičkového detektoru dostaneme odhad zpoždění, viz rovnici 2.17.



Obrázek 2.12: Blokové schéma postupu pro výpočet odhadu zpoždění [14]

$$D = argmax_\tau(R_{x_1x_2}(\tau)) \tag{2.17}$$

Zobecněná křížová korelace (*Generalized cross correlation*), dále jen GCC, je velice užívanou metodou pro lokalizaci zdroje zvuku. GCC dopočítává odhad pomocí křížové spektrální hustoty výkonu (*Cross Power Spectral Density*), dále jen CPSD, a filtrování vstupních signálů. Aplikací Fourierovy transformace na křížovou korelaci vypočítáme CPSD. Zpětně dostaneme korelační funkci aplikací rovnice 2.18.

$$R_{x_1x_2}(\tau) = \int_{-\infty}^{\infty} X_1(f) \cdot X_2^*(f) \cdot e^{j \cdot 2 \cdot \pi \cdot f \cdot t} df = \int_{-\infty}^{\infty} G_{x_1x_2}(f) \cdot e^{j \cdot 2 \cdot \pi \cdot f \cdot t} df \tag{2.18}$$

- $X_1(f)$ je signál $x_1(t)$ převedený do frekvenční domény
- $X_2^*(f)$ je komplexně sdružený signál $x_2(t)$ převedený do frekvenční domény
- $G_{x_1x_2}(f)$ je součin $X_1(f)$ a $X_2^*(f)$

Podle obrázku 2.12 jsou signály $x_1(t)$, $x_2(t)$ filtrovány. Ve frekvenčním spektru to znamená, že filtry stačí vynásobit se signály $x_1(t)$, $x_2(t)$ ve frekvenční doméně. Korelaci s filtrovanými vstupy dostaneme použitím rovnice 2.19.

$$R_{y_1y_2}(\tau) = \int_{-\infty}^{\infty} H_1(f) \cdot H_2^*(f) \cdot G_{x_1x_2}(f) \cdot e^{j \cdot 2 \cdot \pi \cdot f \cdot t} df = \int_{-\infty}^{\infty} \psi_g(f) \cdot G_{x_1x_2}(f) \cdot e^{j \cdot 2 \cdot \pi \cdot f \cdot t} df \tag{2.19}$$

- $H_1(f)$ je filtr signálu $x_1(t)$
- $H_2(f)$ je filtr signálu $x_2(t)$
- $\psi_g(f)$ je součin filtrů $H_1(f)$ a $H_2(f)$, nazývané také jako váhovací funkce

Výsledek korelace $R_{y_1y_2}(\tau)$ je již GCC. Existuje mnoho variant váhovacích funkcí $\psi_g(f)$. Rovnice 2.20, 2.21 a 2.22 jsou definice Rothova procesoru, fázové transformace ($Phase\ Transform$) a vyhlazené koherenční transformace ($Smoothed\ Coherence\ Transform$) [14].

$$\psi_g(f) = \frac{1}{G_{x_1x_1}(f)} \tag{2.20}$$

$$\psi_g(f) = \frac{1}{|G_{x_1x_2}|} \tag{2.21}$$

$$\psi_g(f) = \frac{1}{\sqrt{G_{x_1x_1}(f) \cdot G_{x_2x_2}}} \tag{2.22}$$

2.9 Mikrokontroléry třídy STM32F4x

STM32F4x jsou 32bitové mikrokontroléry založené na architektuře ARM Cortex-M4. Toto jádro poskytuje hardwarovou podporu a instrukční sadu pro digitální zpracování signálu. Součástí jádra je jednotka pro výpočet desetiných čísel, dále jen FPU (*Floating Point Unit*). Mikrokontroléry třídy F4 se mezi sebou liší podle dostupných periferií, rychlostí nebo velikostí pamětí. V tabulce 2.1 jsou vypsány jednotlivé typy, jejich vlastnosti a dostupné periferie pro záznam zvuku.

Mikrokontrolér	F_{CPU} (MHz)	Flash (kB)	RAM (kB)	SAI	DFSDM
STM32F469	180	512 až 2056	384	Ano	Ne
STM32F429	180	512 až 2056	256	Ano	Ne
STM32F427	180	1024 až 2056	256	Ano	Ne
STM32F446	180	256 až 512	128	Ano	Ne
STM32F407	168	512 až 1024	192	Ne	Ne
STM32F405	168	512 až 1024	192	Ne	Ne
STM32F401	84	128 až 512	96	Ne	Ne
STM32F410	100	64 až 128	32	Ne	Ne
STM32F411	100	256 až 512	128	Ne	Ne
STM32F412	100	512 až 1024	256	Ne	Ano
STM32F413	100	1024 až 1536	320	Ne	Ano

Tabulka 2.1: Mikrokontroléry STM32F4x a jejich vlastnosti [15]

- SAI (*Sound Audio Interface*) je periferie, která se specializuje na audiodigitální data
- DFSDM (*Digital Filter for Sigma – Delta Modulators*) je periferie, která poskytuje digitální filtraci signálů pomocí sigma-delta modulátoru

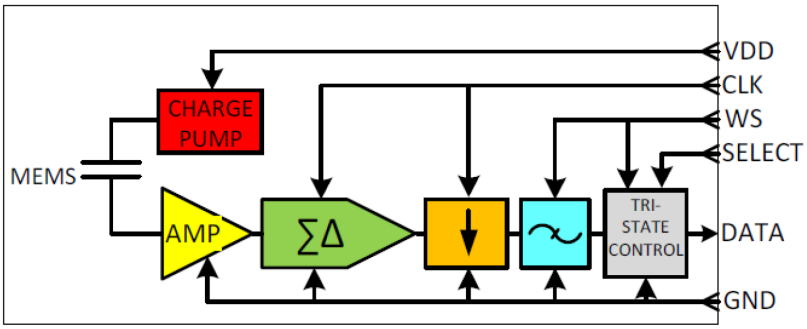
Jádro M4 poskytuje dostatečný výpočetní výkon pro trénování a inferenci neuronových sítí menšího rozsahu, jako jsou konvoluční nebo rekurentní neuronové sítě. Při implementaci neuronových sítí je nutné optimalizovat model sítě a využívat různé techniky komprese a kvantizace, aby se dosáhlo dostatečné efektivity a snížení paměťového a výpočetního nároku. Pro mikrokontroléry STM32 existuje specializovaná knihovna STM32Cube.AI, která usnadňuje implementaci neuronových sítí [15].

Kapitola 3
Použitý hardware

3.1 Mikrofon

Zvuk je nahráván pomocí mikrofonu SPH0645LM4H. Jedná se o MEMS digitální mikrofon, který obsahuje akustický senzor, sigma-delta ADC a rozhraní pro úpravu signálu na průmyslový 24bitový I^2S standard.

Blokový diagram 3.1 znázorňuje fungování mikrofonu. Nábojová pumpa nastaví pracovní bod pro senzor. Přes zesilovač se získaný analogový signál převede na PDM (*Pulse Density Modulation*) data pomocí sigma-delta převodníku. Digitální zvuk v mikrofonu projde decimačním filtrem s faktorem 64 a nízkofrekvenčním filtrem. Poslední blok je řízení třístavového výstupu, které určuje na základě signálu WS a Select, jestli je datový pin ve vysoké impedanci, nebo zda vysílá signál. To umožňuje připojit dva mikrofony na stejný I^2S port. Když jsou pin Select a signál WS v logické 1, datový pin vysílá data. Analogicky pro logickou 0.



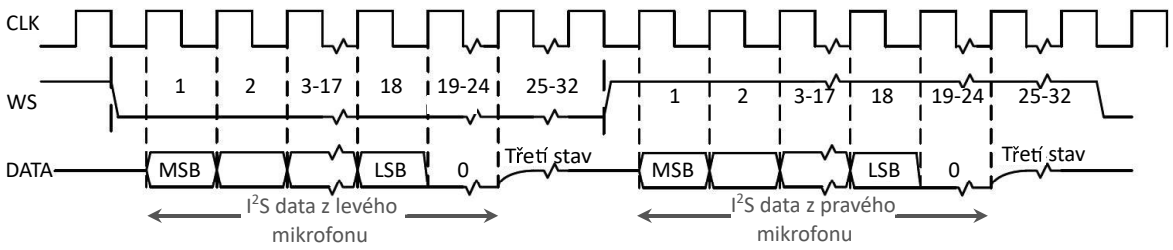
Obrázek 3.1: Blokový diagram SPH0645LM4H [16]

Pro správnou funkci mikrofonu musí master posílat dva hodinové signály BCLK a WS. Na signálu BCLK operuje delta-sigma převodník a decimační filtr. Může se pohybovat v rozmezí 1,024 MHz až 4,096 MHz. Hodinový signál WS reprezentuje vzorkovaná data. Mezi signály BCLK a WS platí vztah z rovnice 3.1.

$$f_{\mathrm{WS}} = \frac{f_{\mathrm{BCLK}}}{N_{Dec}} \tag{3.1}$$

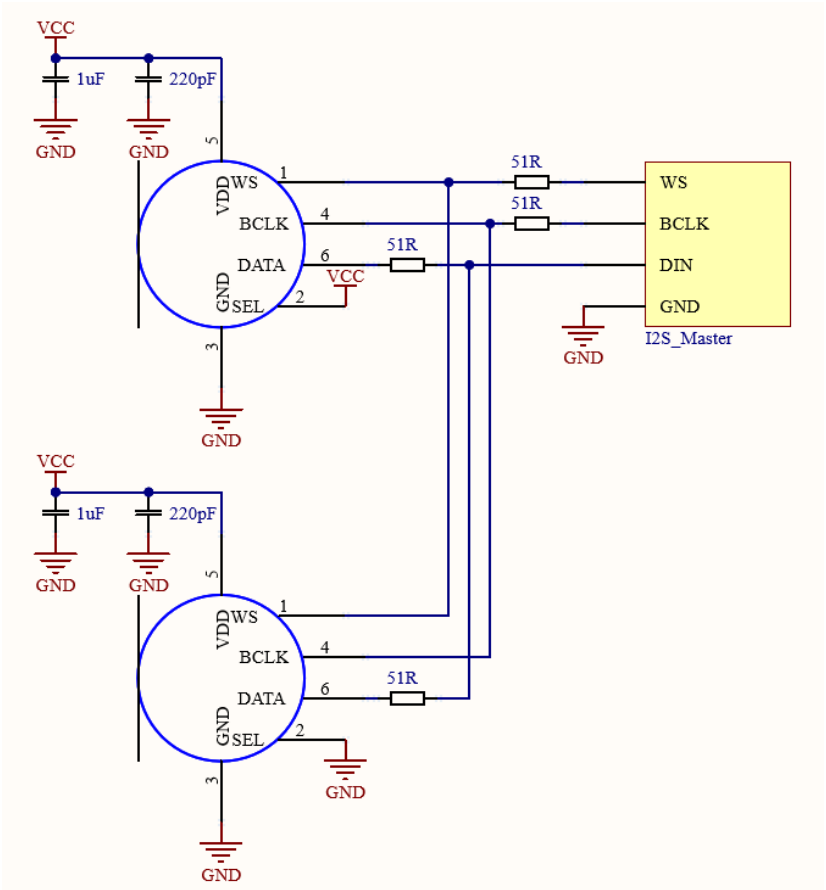
- f_{WS} je frekvence hodinového signálu WS
- f_{BCLK} je frekvence hodinového signálu BCLK

- N_{Dec} je decimační faktor, který je u mikrofonu SPH0645LM4H nastaven na 64



Obrázek 3.2: Časový diagram mikrofonu SPH0645LM4H [16]

Datový formát výstupu je 24bitový I²S standard s dvojkovým doplňkem a MSB se nachází na první pozici. Přesnost mikrofonu je pouze 18 bitů, nepoužité bity jsou nulové.



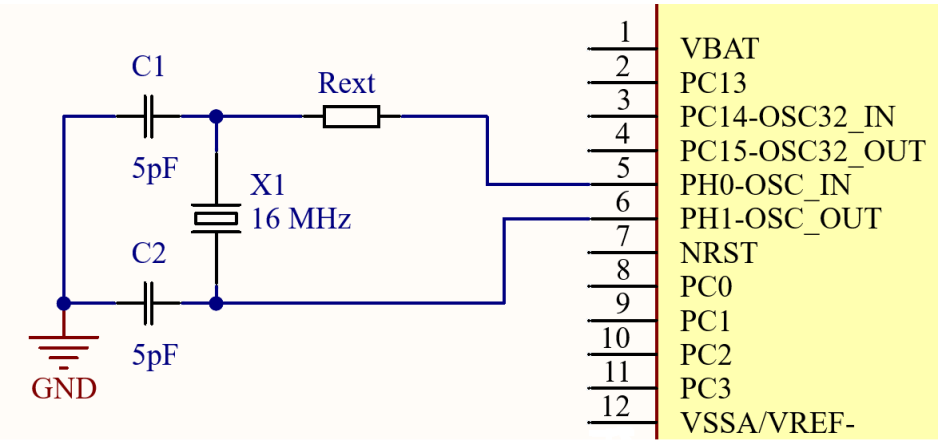
Obrázek 3.3: Doporučené zapojení SPH0645LM4H [16]

Pomocí decimačního filtru se PDM data převedou na PCM (*Pulse Code Modulation*) formát, který je shodný s formátem trénovacích dat. Proto byl tento mikrofon vybrán. Dalšími možnostmi konverze z PDM na PCM je použít softwarovou knihovnu nebo další hardwarovou komponentu. Konverze pomocí softwaru není možná. Vstupní data by se zvýšila 64krát a zároveň by bylo plýtváno výpočetním časem [16].

3.2 Mikrokontrolér

Nahraný zvuk zpracovává a vyhodnocuje mikrokontrolér STM32F411RE. Jedná se o 32bitový mikrokontrolér s M4 jádrem, které pracuje až na 100 MHz. Jádro obsahuje FPU a DSP instrukce. STM32F411RE je vybaven 512kB pamětí flash a 128kB pamětí SRAM. Nahrávat kód do paměti lze pomocí JTAG nebo SWD portu, pro které je potřeba programátor.

Při restartu je pro hodiny CPU vybrán 16MHz vnitřní RC oscilátor s 1% přesností. Softwarově lze vybrat externí krystalový nebo keramický oscilátor v rozmezí 4 MHz až 26 MHz. Jeho správné zapojení je znázorněno na obrázku 3.4.



Obrázek 3.4: Doporučené zapojení externího oscilátoru k STM32F411 [17]

Mikrokontrolér je vybaven pěti periferiemi I^2S, které jsou multiplexovány s SPI. Mohou operovat v roli master nebo slave a pracovat v 16bitovém nebo 32bitovém rozlišení. Nahrávané audio podporuje vzorkovací frekvenci od 8 kHz do 192 kHz. Mikrokontrolér je vybaven fázovým závěsem pro audio, dále jen PLLI2S (*Phase Locked Loop*). Díky němu je hodinový signál I^2S přesný a zároveň se nemusí měnit rychlost CPU. Periferii je možné připojit k DMA a tak umožnit nepřetržité nahrávání audio dat do paměti bez nutnosti přerušovat chod celého programu [17].

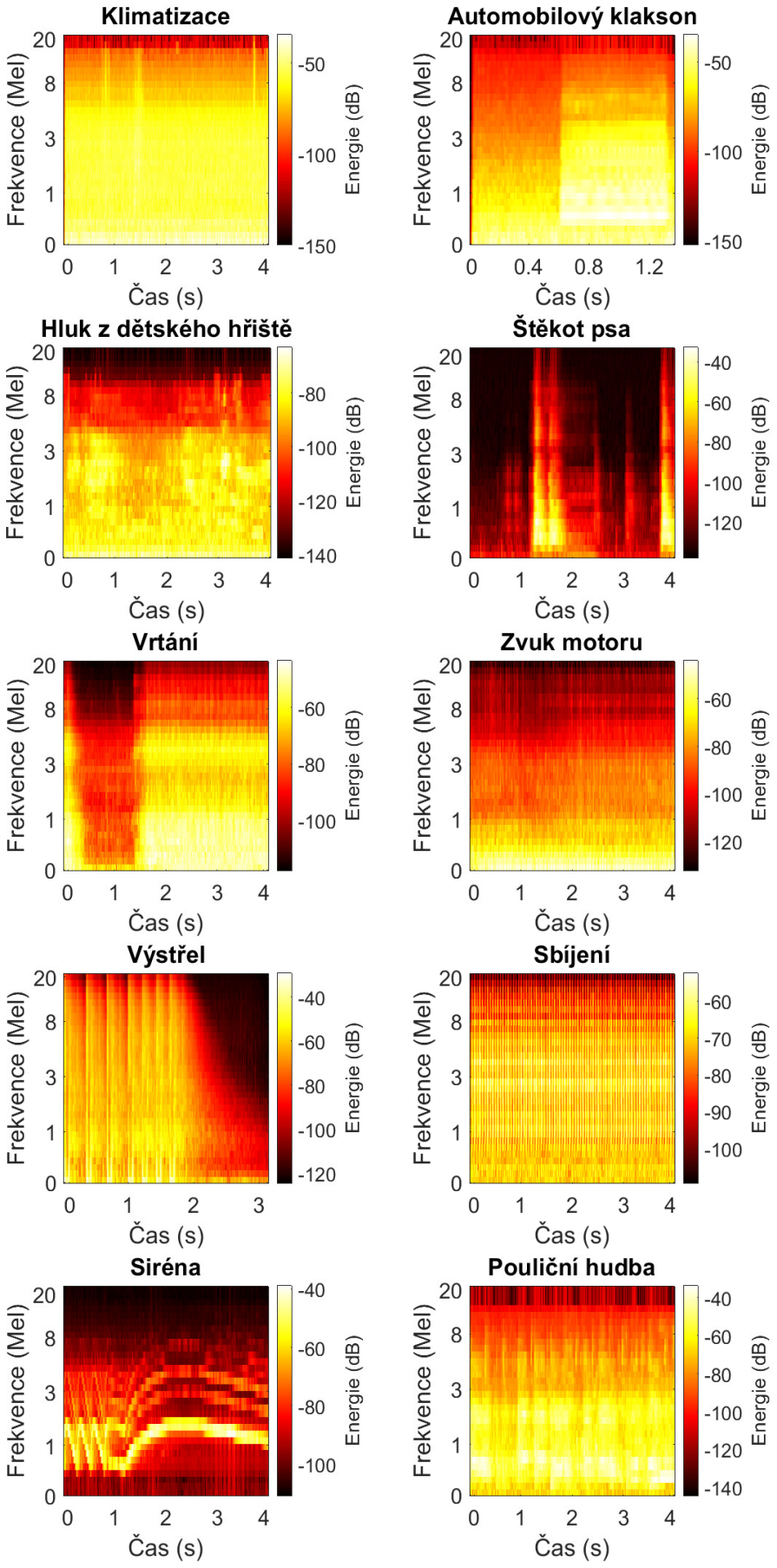
Kapitola 4
Použitý software

4.1 Dataset

Dataset UrbanSound8K je použit k trénování a validaci modelu. Obsahuje 8732 označených zvukový stop do čtyř sekund z urbanistického prostředí. Zvukové záznamy jsou rozděleny do deseti různých složek a jsou připraveny k desetinásobné křížové validaci. Vzorkovací frekvence, rozlišení a počet kanálů se mezi jednotlivými záznami liší. K datasetu jsou přiložena meta-data ve formátu .csv [18]. V tabulce 4.1 je seznam tříd a jejich přiřazená ID. Na obrázku 4.1 jsou grafické reprezentace jednotlivých tříd v Melově škále.

Tabulka 4.1: Třídy zvuků a k nim přiřazená ID

Název třídy	Přiražené ID
Klimatizace	0
Automobilový klakson	1
Zvuk z dětského hřiště	2
Štěkající pes	3
Vrtačka	4
Motor	5
Výstřel ze zbraně	6
Sbíječka	7
Siréna	8
Pouliční muzika	9



Obrázek 4.1: Spektrogramy jednotlivých tříd v Melově škále

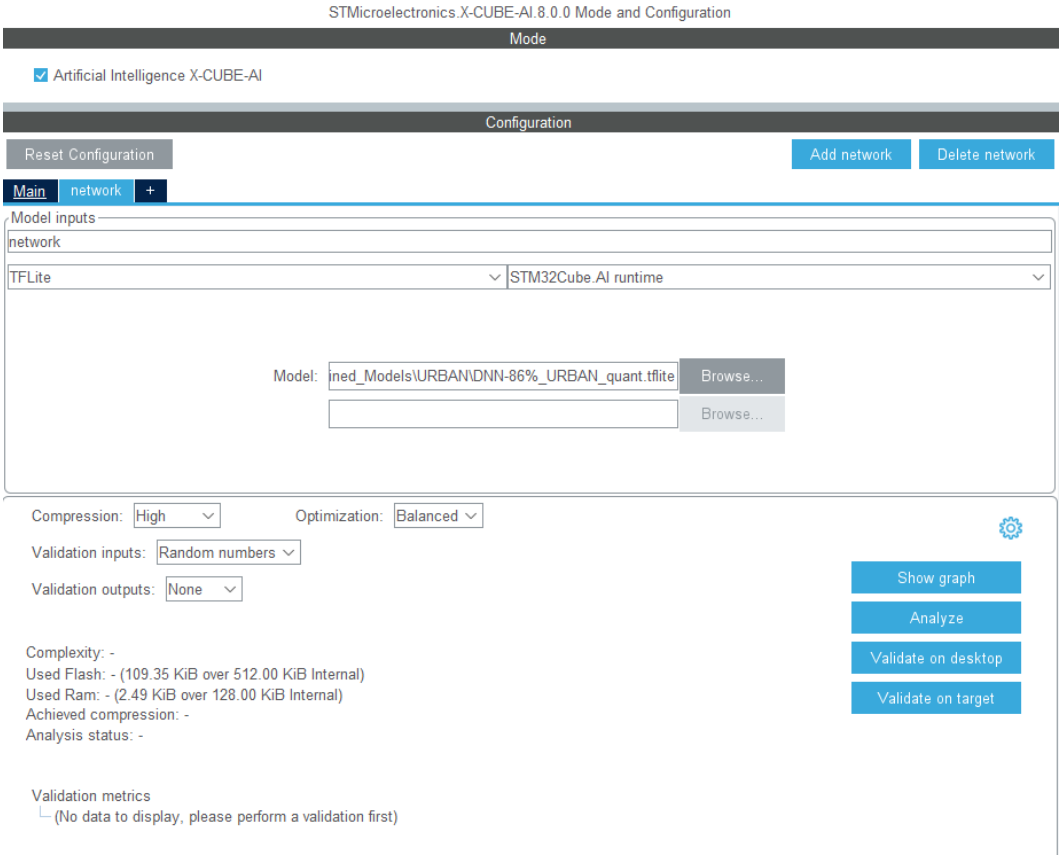
4.2 Prostředí pro trénování modelu

K trénování modelu neuronové sítě je použit TensorFlow a Keras v prostředí Google Colab. Jedná se o cloudový Jupyter notebook v prohlížeči, který nabízí přístup k výkonným GPU NVIDIA V100. Předzpracování využívá arm-cmsis knihovnu v jazyce Python, která je součástí CMSIS-DSP repositáře [19].

4.3 Implementace modelu na mikrokontrolér

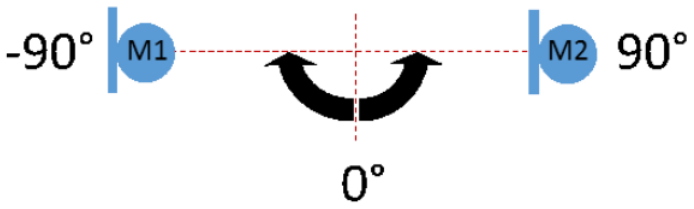
Mikrokontrolér byl naprogramován v STM32CubeIDE. Integrované prostředí má zabudované pomocné GUI STM32CubeMX, které umožňuje graficky nastavovat jednotlivé periferie a stahovat některé softwarové knihovny. Při práci s tímto nástrojem je možné si zvolit, zda bude vygenerovaný kód používat knihovnu HAL nebo LL (*Low Layer*). V této práci jsou použity tři softwarové knihovny, X-Cube-AI (verze 8.0.0), AcousticSL a CMSIS-DSP.

X-Cube-AI je rozšiřující knihovna, která je součástí ekosystému STM32Cube.AI. Umožňuje převádět předtrénované modely strojového učení a neuronových sítí. Lze jej využívat v STM32CubeMX, stačí tedy nahrát předtrénovaný model do grafického nástroje, vybrat vhodnou optimalizaci a analyzovat model. Po analýze dostaneme informaci, zda lze model implementovat do mikrokontroléru, počet MACC (*Multiple Accumulate operations*) operací, požadovanou velikost paměti flash a SRAM. Balíček podporuje konverzi natrénovaných modelů z Keras, ONNX, TensorFlowLite [20].

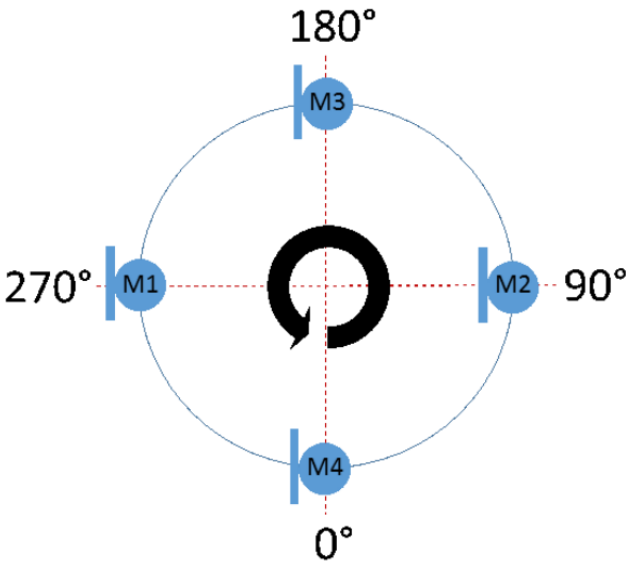


Obrázek 4.2: Aplikace STM32CubeMX s knihovnou X-Cube-AI [20]

AcousticSL je knihovna, která implementuje tři algoritmy pro lokalizaci zvuku: Křížová korelace v časové doméně, zobecněná křížová korelace s fázovou transformací a algoritmus BMPH (*Block Matching Pursuit with Hangover*). Odhad je dopočítán ze dvou nebo čtyř mikrofonů. V inicializační funkci knihovny se nastavují parametry jako je vzdálenost mikrofonů od sebe, počet kanálů, počet mikrofonů se sdíleným bufferem a výběr algoritmu. Vzdálenost mikrofonů se udává v desetinách milimetrů a jedná se o vzdálenost mezi mikrofonem M_1M_2 a M_3M_4, viz obrázky 4.3 a 4.4 [21].



Obrázek 4.3: Rozložení se dvěma mikrofony [21]



Obrázek 4.4: Rozložení se čtyřmi mikrofony [21]

CMSIS (*Common Microcontroller Software Interface Standard*) je softwarová abstrakční vrstva pro procesory s architekturou ARM Cortex-M a poskytuje standardizovaný přístup k hardwarovému a softwarovému rozhraní. Jeho výhodou je zjednodušení softwaru pro Cortex-M mikrokontroléry a usnadňuje přenositelnost softwaru. Knihovna DSP je součástí této vrstvy a obsahuje funkce pro digitální zpracování signálu. Dokáže pracovat s 8bitovými, 16bitovými a 32bitovými celými čísly a s 32bitovými desetinnými čísly. Knihovna je optimalizovaná pro SIMD (*Single Instruction Multiple Data*) instrukční sadu, která je dostupná pro jádra M4, M7, M33 a M35P [22].

CMSIS-DSP knihovna obsahuje potřebné hlavičkové a zdrojové soubory. Zároveň v ní najdeme i skripty v jazyce Python, které slouží k vygenerování konfiguračního souboru pro počítání MFCC [19].

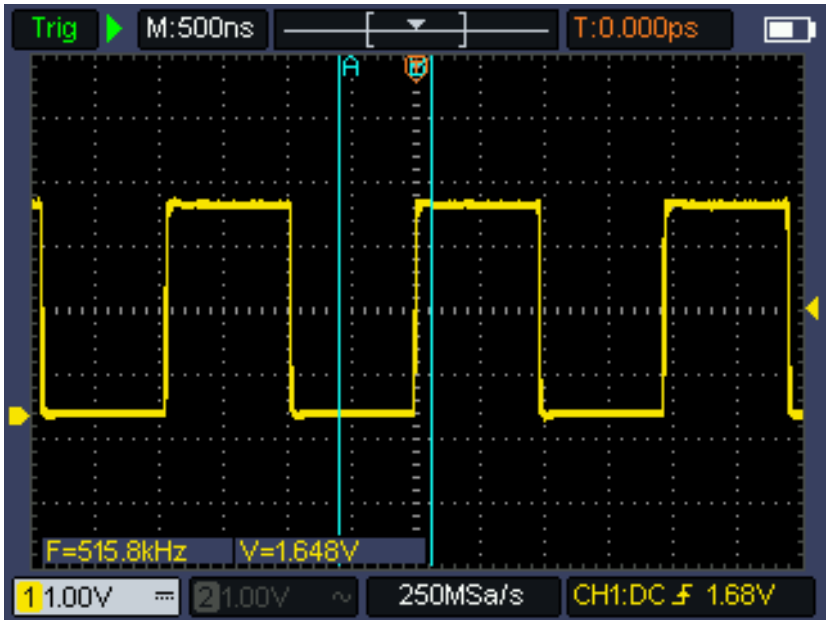
Kapitola 5
Metody

5.1 Zaznamenání zvuku

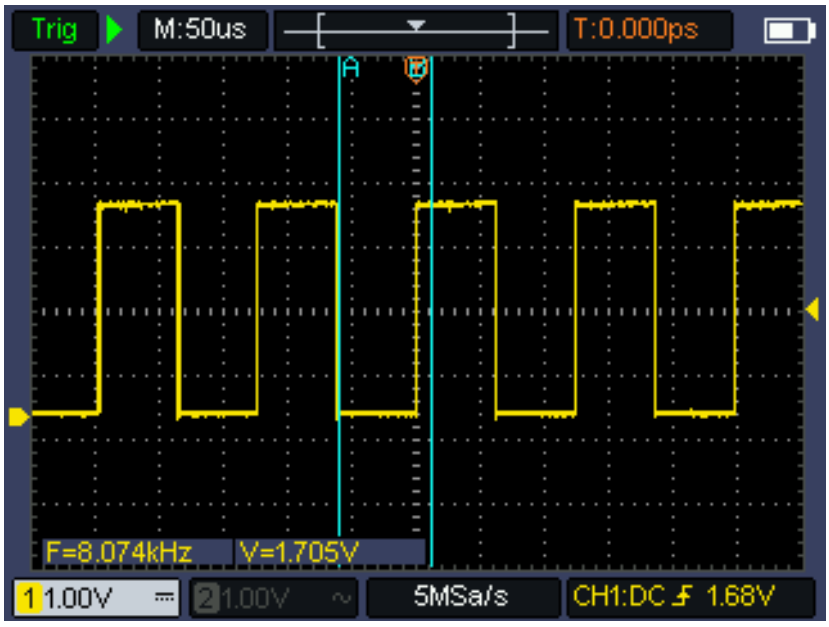
Zvuk je zaznamenán pomocí čtyř mikrofonů SPH0645LM4H popsaných na obrázku 3.1. Jsou po dvojicích připojeny k MCU a komunikují přes I²S Philips standard. Vzorkovací frekvence nahrávaného zvuku je 8 kHz pro každý mikrofon. Data jsou pro každou periferii uložena v bufferu o velikosti 512 v 16bitovém celočíselném formátu. Tato velikost odpovídá 32 ms nahraných dat. Hodinový signál I²S je nastaven na 153,6 MHz pomocí registru RCC_PLLI2SCFGR [23]. Hodnota M je nastavena na 10, N na 192 a R na 2. Výpočet hodinového signálu získáme aplikací rovnice 5.1. Záznam dat provádí DMA v kruhovém režimu. Po každých 256 půlslovech ($halfword$) je vyvoláno přerušení a následuje předzpracování nahraného vzorku.

$$f_{\mathrm{I^2S}} = \frac{f_{\mathrm{PLL\_Source}} \cdot N}{M \cdot R} \tag{5.1}$$

- $f_{\mathrm{PLL\_Source}}$ je frekvence zdroje pro fázový závěs



Obrázek 5.1: Záznam signálu CLK z periferie I²S



Obrázek 5.2: Záznam signálu WS z periferie I²S

5.2 Předzpracování signálu

Natrénovaná neuronová síť přijímá vstup v podobě matice obsahující MFCC. Matice obsahuje 61 vektorů s 16 kepstrálními koeficienty. Pro jejich výpočet je použita knihovna CMSIS-DSP, která je dostupná i v jazyce Python. Nejprve je potřeba si nadefinovat datový typ, vzorkovací kmitočet, délku Fourierovy transformace, počet kepstrálních koeficientů, minimální a maximální frekvenci, počet filtrů ve filtrační bance a druh váhovacího okna. Délka váhovacího okna musí být stejná jako délka Fourierovy transformace. Funkce $mfcc.melFilterMatrix$ dopočítá jednotlivé filtry. Výsledkem funkce jsou matice s filtračními hodnotami, pozicemi a délkami jednotlivých filtrů. Posledním krokem je výpočet matice s koeficienty DCT pomocí funkce $mfcc.dctMatrix$. Výpočet MFCC se provede inicializací struktury pomocí funkce $dsp.arm\_mfcc\_init\_f32$. Na každých 32 ms audiozáznamu se aplikuje funkce $dsp.arm\_mfcc\_f32$, která vrátí vypočítaný vektor s 16 kepstrálními koeficienty. Této funkci se přidává ukazatel na dočasný buffer, který funkce využívá pro vlastní výpočty. Dočasný buffer musí mít velikost o dva větší, než je délka Fourierovy transformace.

Tabulka 5.1: Nastavené parametry pro výpočet MFCC

Parametr	Hodnota
Vzorkovací kmitočet	16 kHz
Délka FFT	512
Počet MFCC	16
Minimální frekvence	64 Hz
Maximální frekvence	8 kHz
Počet filtračních bank	20
Druh váhovacího okénka	Hamming
Datový typ	32bitové desetinné číslo

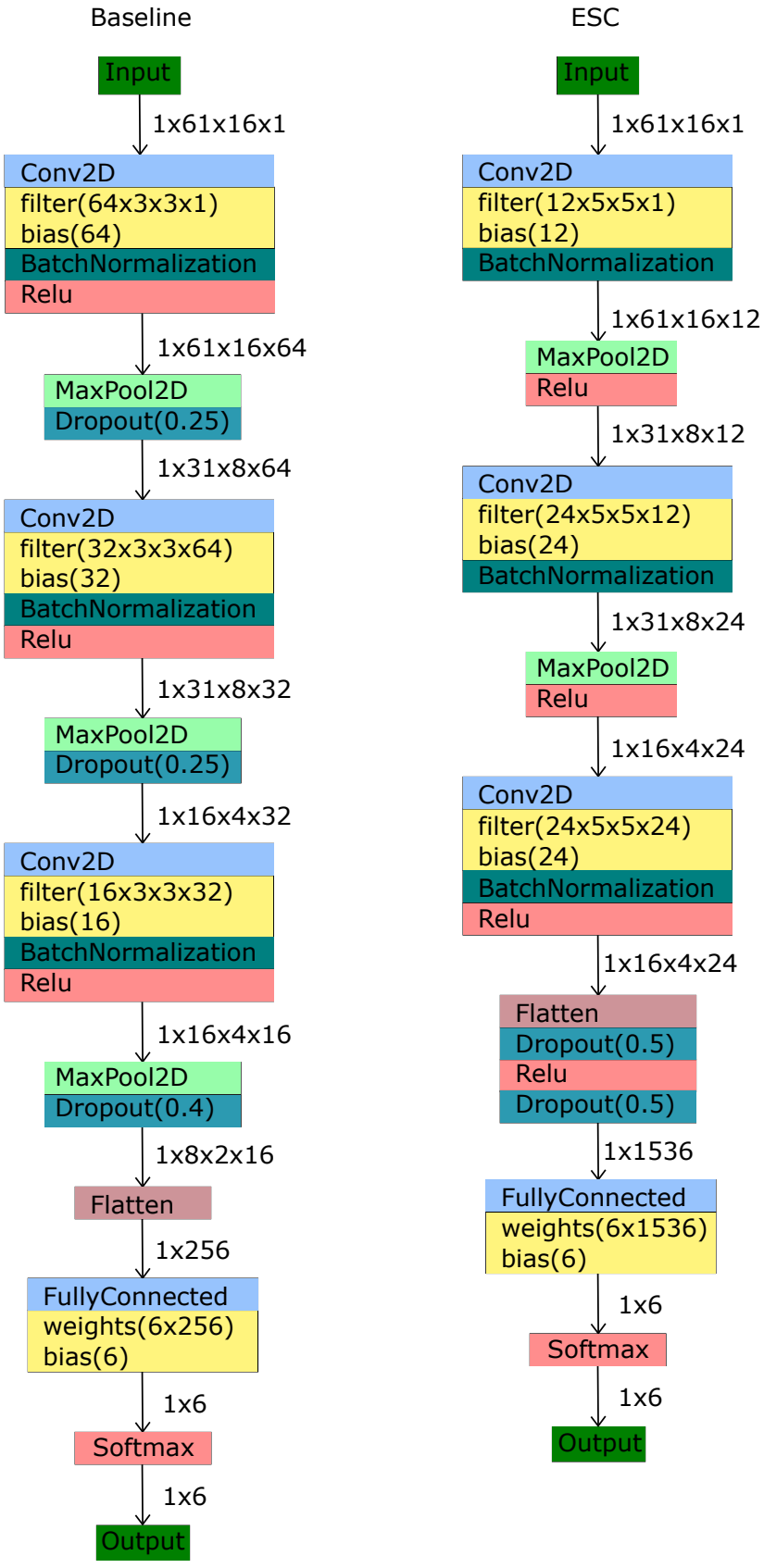
Pro mikrokontrolér se postup mírně liší. Matice s koeficienty DCT, filtračními hodnotami, váhovacím okénkem, pozicemi a délkami jednotlivých filtrů je vypočítána již v jazyce Python pomocí skriptu uloženém v github repositáři [19]. Skript je umístěn v Scripts\GenMFCCDataForCPP.py. Skriptu je potřeba při spuštění předat konfigurační soubor .yaml, ve kterém jsou nadefinovány všechny důležité parametry. Skript vytvoří hlavičkový a zdrojový soubor pro mikrokontrolér. Soubory stačí přidat do projektu mikrokontroléru.

```yaml
dct:
  config1_f32:
    melFilters: 20
    dctOutputs: 13
    type: "f32"

melfilter:
  config1_f32:
    fftlength: 512
    fmin: 64
    fmax: 8000
    samplingRate : 16000
    melFilters: 20
    type: "f32"

window:
  config1_f32:
    fftlength: 512
    type: "f32"
    win: "hamming"
```

Obrázek 5.3: Ukázka konfiguračního souboru pro vygenerování hlavičkového a zdrojového souboru pro mikrokontrolér

5.3 Trénování modelu

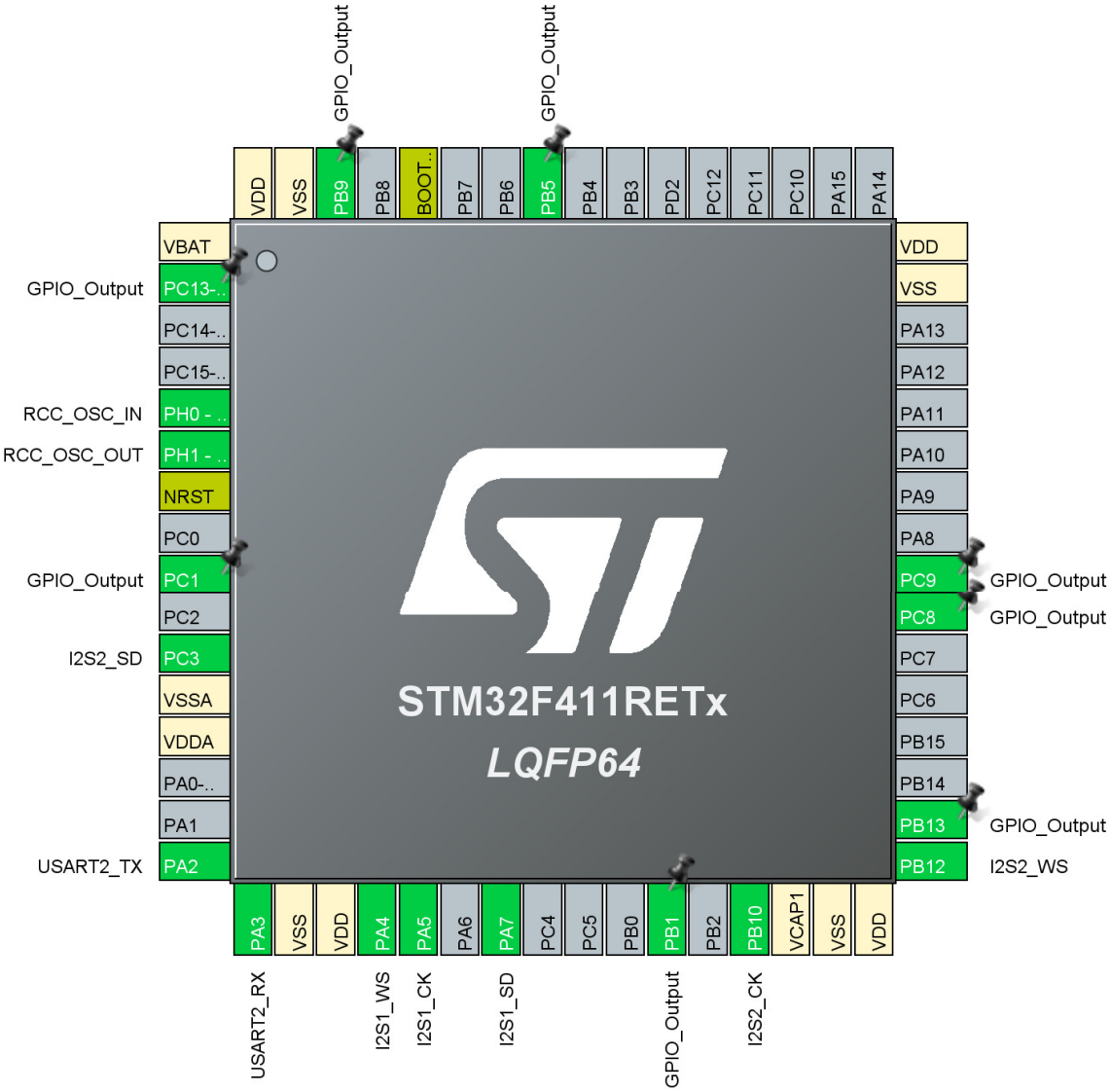
Modely jsou natrénovány pomocí platformy Keras. Dataset je rozdělen na trénovací a validační skupinu dvěma způsoby. První způsob je náhodné rozdělení v poměru 8:2. Druhým způsobem je využití 10násobné křížové validace. Použité modely jsou malé konvoluční neuronové sítě, viz obrázek 5.4. Pro trénování je zvolen optimalizační algoritmus ADAM spolu s řídkou kategoriální křížovou entropií. Rychlost učení je měněna během trénování každých 40 epoch. U vyššího počtu epoch nedochází k zlepšení na testovacích i validačních datech. Celkem jsou modely trénovány na 120 epochách. Rychlost učení je v jednotlivých etapách je 0,01, 0,001 a 0,0001. Batch size je nastaven na 32. Natrénovaný model je převeden do formátu .tflite a následně je nahrán promocí X-CUBE-AI do mikrokontroléru. Z datasetu jsou vypuštěny labely pouliční hudby, výstřelu, hluku z dětského hřiště a psího štěkotu. Modely, které nejsou natrénovány na těchto labelech dosahují mnohem vyšší přesnosti při rozpoznávání sirény v reálném čase na mikrokontroléru. Na obrázku 5.4 jsou architektury dvou natrénovaných a použitých modelů se jmény Baseline a ESC.



Obrázek 5.4: Schéma modelů natrénovaných neuronových sítí

5.4 Program v mikrokontroléru

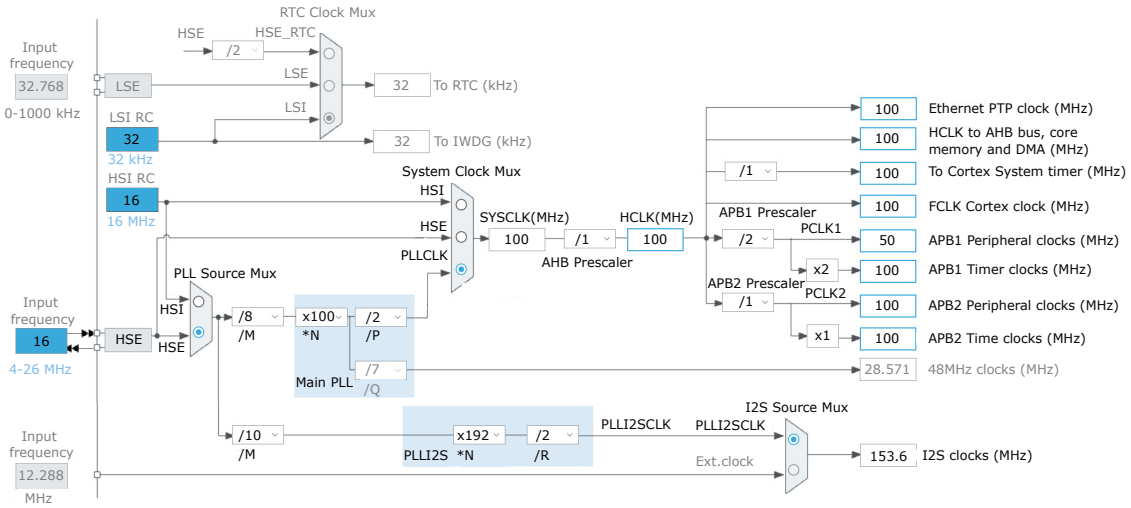
Konečným výstupem mikrokontroléru je reagování na rozpoznání sirény a rozsvícení LED ve směru zdroje zvuku. Prvním krokem je inicializace všech potřebných periferií, viz obrázek 5.5 a tabulka 5.2, a nastavení hodinového signálu, viz obrázek 5.6. Automatické generování inicializací s knihovnou HAL je provedeno pomocí GUI STM32CubeMX [24].



Obrázek 5.5: Nastavení pinů na STM32F411 v STM32CubeMX

Tabulka 5.2: Seznam použitých pinů

Název pinu	Funkce
PB1	GPIO výstup
PB5	GPIO výstup
PB9	GPIO výstup
PB13	GPIO výstup
PC1	GPIO výstup
PC8	GPIO výstup
PC9	GPIO výstup
PC13	GPIO výstup
PA4	I^2S1 WS
PA5	I^2S1 CK
PA7	I^2S1 SD
PB12	I^2S2 WS
PB10	I^2S2 CK
PC3	I^2S2 SD
PA2	USART2 TX
PA3	USART2 RX



Obrázek 5.6: Nastavení hodinových signálů pro STM32F411 v STM32CubeMX

Potřeba je také nahrát do projektu knihovny CMSIS-DSP, STM32_AcousticSL a vygenerovaný zdrojový a hlavičkový kód pro výpočet MFCC zmíněný v kapitole 5.2. Pro jejich správné fungování je nezbytné aktivovat FPU (*Floating Point Unit*) zapsáním logické 1 do bitů 20-23 v registru CPACR, viz obrázek 5.7 [25].

```
static void FPU_Enable(void)
{
    SCB->CPACR |= (1U<<20);
    SCB->CPACR |= (1U<<21);
    SCB->CPACR |= (1U<<22);
    SCB->CPACR |= (1U<<23);
}
```

Obrázek 5.7: Funkce pro aktivaci FPU

Před použitím knihoven je potřeba nainicializovat jejich struktury. Na obrázcích 5.8 a 5.9 jsou ukázány inicializační funkce pro výpočet MFCC a model neuronové sítě. Na obrázku 5.10 je ukázka konfigurační struktury pro lokalizaci zvuku a na obrázku 5.11 je definice inicializační funkce, ve které je potřeba nastavit požadované parametry.

```
arm_status arm_mfcc_init_f32(
  arm_mfcc_instance_f32 * S,
  uint32_t fftLen,
  uint32_t nbMelFilters,
  uint32_t nbDctOutputs,
  const float32_t *dctCoefs,
  const uint32_t *filterPos,
  const uint32_t *filterLengths,
  const float32_t *filterCoefs,
  const float32_t *windowCoefs
  );
```

Obrázek 5.8: Inicializační funkce pro výpočet MFCC

arm_mfcc_instance_f32 *S; Ukazatel na mfcc strukturu. Funkce nahraje všechna potřebná data do struktury. Slouží jako výstup.

uint32_t fftlen; Velikost rychlé Fourierovy transformace. Hodnota je shodná z velikostí váhovacího okénka. Nastaveno na 512.

uint32_t nbMelFilters; Počet použitých filtračních bank k vypočtení kepstrálních koeficientů. Nastaveno na 20.

uint32_t nbDctOutputs; Počet výstupních MFCC. Nastaveno na 16.

const float32_t *dctCoefs; Ukazatel na transformační matici s DCT koeficienty.

const uint32_t *filterPos; Ukazatel na pole s pozicemi jednotlivých filtrů.

const uint32_t *filterLengths; Ukazatel na pole s délkami jednotlivých filtrů.

const float32_t *filterCoefs; Ukazatel na pole s filtračními koeficienty.

const float32_t *windowCoefs; Ukazatel na matici obsahující hodnoty Hammingova okénka.

```
ai_error ai_network_create_and_init(
  ai_handle* network,
  const ai_handle activations[],
  const ai_handle weights[]
);
```

Obrázek 5.9: Inicializační funkce pro model neuronové sítě

ai_handle *network; Ukazatel, který knihovna využívá pro své výpočty. Při inicializaci má ukazatel hodnotu 0.

const ai_handle activations[]; Konstantní pole s velikostí rovnu maximálnímu odhadu využití paměti RAM. Knihovna do tohoto pole ukládá všechny mezivýpočty při inferenci.

const ai_handle weights[]; Konstantní pole s předtrénovanými váhami. Slouží při trénování neuronové sítě na mikrokontroléru. Ukazatel na toto pole je roven 0, funkce není v práci využita.

```
typedef struct
{
  uint32_t algorithm;
  uint32_t sampling_frequency;
  uint32_t channel_number;
  uint8_t ptr_M1_channels;
  uint8_t ptr_M2_channels;
  uint8_t ptr_M3_channels;
  uint8_t ptr_M4_channels;
  uint16_t M12_distance;
  uint16_t M34_distance;
  uint32_t internal_memory_size;
  uint32_t *pInternalMemory;
  int16_t samples_to_process;
} AcousticSL_Handler_t;
typedef struct
{
  uint16_t threshold;
  uint32_t resolution;
} AcousticSL_Config_t;
```

Obrázek 5.10: Konfigurační struktura pro lokalizaci zvuku

```
void AcousticParams_Init(void)
{
    __IO uint32_t error_value = 0;

    /* Enable CRC peripheral to unlock the library */
    __CRC_CLK_ENABLE();

    /*Setup Source Localization static parameters*/
    hacoust1.channel_number = AUDIO_IN_CHANNELS;
    // Distance is in decimals of millimeters
    hacoust1.M12_distance = 900;
    hacoust1.M34_distance = 900;
    hacoust1.sampling_frequency = AUDIO_IN_SAMPLING_FREQ;
    hacoust1.algorithm = ACOUSTIC_SL_ALGORITHM_GCCP;
    hacoust1.ptr_M1_channels = 2;
    hacoust1.ptr_M2_channels = 2;
    hacoust1.ptr_M3_channels = 2;
    hacoust1.ptr_M4_channels = 2;
    hacoust1.samples_to_process = SAMPLES_2_PROCESS;
    AcousticSL_getMemorySize(&hacoust1);

    hacoust1.pInternalMemory =
            (uint32_t*)malloc(hacoust1.internal_memory_size);
    AcousticSL_Init(&hacoust1);

    /*Setup Source Localization dynamic parameters*/
    hconfacoust1.resolution = 6;
    hconfacoust1.threshold = 36;
    AcousticSL_setConfig(&hacoust1, &hconfacoust1);

}
```

Obrázek 5.11: Inicializační funkce pro lokalizaci zdroje zvuku

uint32_t algorithm; Zvolený algoritmus pro lokalizaci zvuku. Lze vybrat mezi BMPH ($Block-Matching Pursuit with Hangover$), GCC-PHAT ($Generalized\ Cross\ Correlation\ with\ Phase\ Transform$) nebo křížovou korelací. Zvoleno GCC-PHAT nastavením proměnné na 2.

uint32_t sampling_frequency; Vzorkovací frekvence mikrofonů. Nastavena na 8000.

uint32_t channel_number; Počet použitých mikrofonů. Pro 360° lokalizaci zdroje zvuku jsou použity 4.

uint8_t ptr_Mx_channels; Počet mikrofonů sdílejících buffer s mikrofonem x. Mikrokontrolér využívá dva mikrofony na jedné periferii I^2S. Hodnota pro všechny čtyři parametry je rovna 2.

uint16_t Mxy_distance; Vzdálenost mezi mikrofonem x a y v desetinách milimetru.

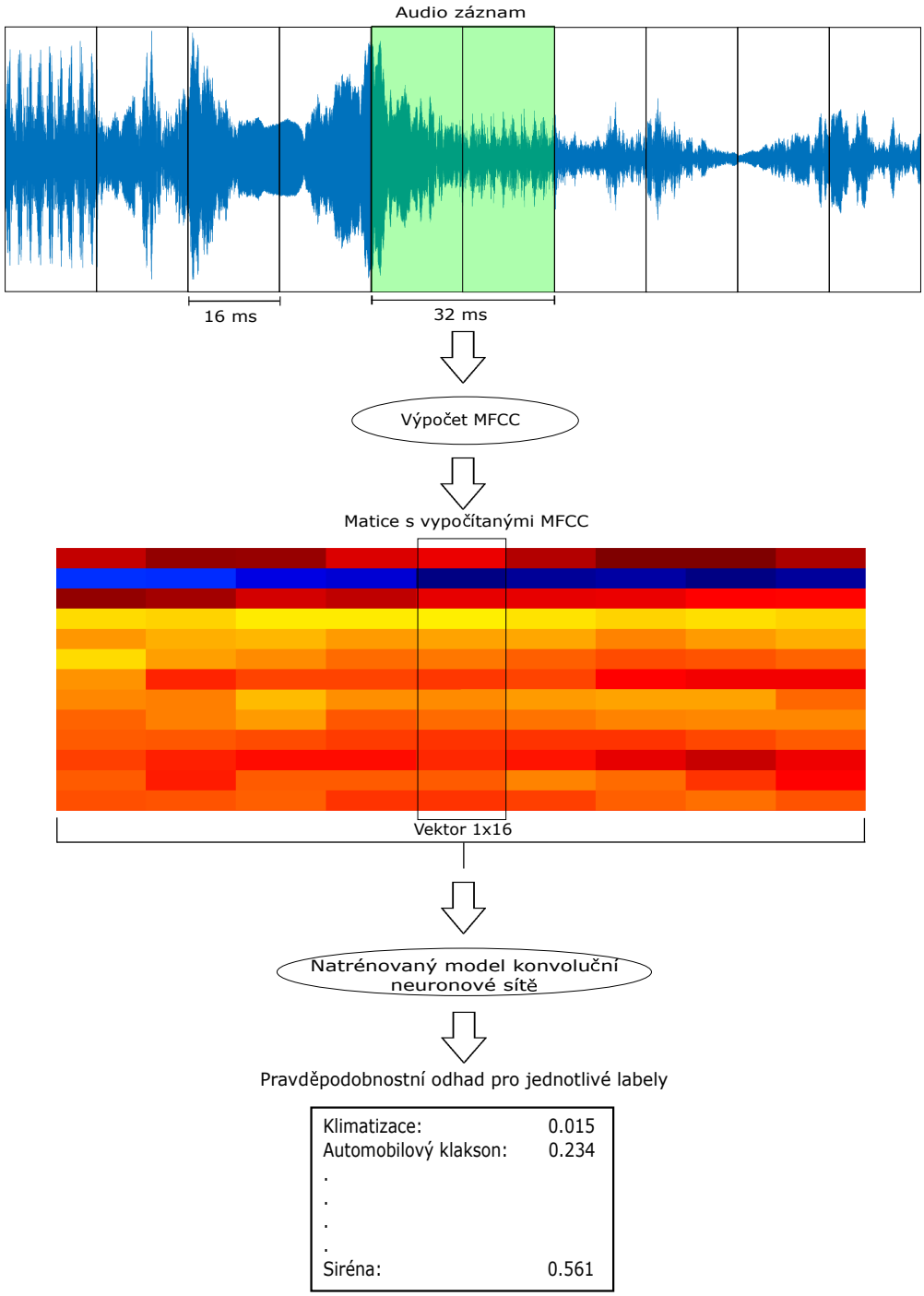
Rozložení mikrofonu je vidět na obrázku 4.4. Nastaveno na 900.

uint32_t internal_memory_size; Potřebná paměť k fungování knihovny. V inicializační funkci je vypočítána pomocí funkce $AcousticSL\_getMemorySize(\&hacoust1)$.

uint32_t *pInternalMemory; Ukazatel na alokované pole jehož velikost je uložena v proměnné *internal_memory_size*. Paměť je alokována pomocí funkce malloc.

int16_t samples_to_process; Počet vzorků zpracovaných při jednom volání knihovny. Nastaveno na 256.

Program lze celkově rozdělit na klasifikační a lokalizační část. Klasifikační část začíná přerušením vyvolaným periferií I^2S. Pro klasifikaci jsou využity pouze dva mikrofony připojené k I^2S1, ze kterých je odebrán audio záznam o délce 32 ms. Z tohoto záznamu je vypočítán vektor s 16 kepstrálními koeficienty, který je vložen na konec matice obsahující předchozí vektory. Matice pracuje na principu FIFO (*First In First Out*). Analyzující okénko bufferu se posune o 16 ms a program opakuje předchozí postup. Po patnácti nově vypočítaných vektorech se matice 61x16 vloží do vstupu modelu neuronové sítě. Je provedena klasifikace, jejíž výstupem je pravděpodobnostní odhad. Algoritmus je znázorněn na obrázku 5.12.



Obrázek 5.12: Přehled předzpracování a klasifikace zvuku

Jak bylo řečeno v kapitole 5.2, data jsou ukládána pomocí DMA do 16bitového bufferu o velikosti 512, což odpovídá 32 ms dat. Matici s kepstrálními koeficienty nahrazuje v programu kruhová fronta. Její definice je spolu s přidruženými funkcemi zobrazena na obrázku 5.13.

```
typedef struct
{
    ai_float data[MFCC_DATA_SIZE];
    uint8_t head;
    uint8_t tail;
    uint8_t filled;
    uint8_t vec_count;
} CircularQueue;

void initCircularQueue(CircularQueue* q);
void enqueueCircular(CircularQueue* q, float* data);
void dequeueCircular(CircularQueue* q);
void addData(CircularQueue* q, float* data);
void takeData(CircularQueue* q, float* output);
```

Obrázek 5.13: Struktura kruhové fronty a její přidružené funkce

ai_float data; Datové pole reprezentuje kruhovou frontu. Jeho velikost je 61x16 a reprezentuje matici pro kepstrální koeficienty.

uint8_t head; Index head reprezentuje první vektor v matici s kepstrálními koeficienty. Jedná se také o vektor, který je přepsán nově příchozím vektorem. Hodnota indexu se pohybuje v rozmezí od 0 do 61 a při interakci s daty je násoben 16, aby ukázal na správný prvek v poli.

uint8_t tail; Index tail reprezentuje poslední vektor v matici s kepstrálními koeficienty. Tento vektor reprezentuje poslední přidaný vektor do matice. Interakce s daty a rozmezí je stejné jako u indexu head.

uint8_t filled; Proměnná *filled* reprezentuje informaci, zda byla matice alespoň jednou naplněna. Algoritmus naplnění se lehce liší v závislosti na této proměnné. Nabývá hodnoty 0 pouze před prvním naplněním.

uint8_t vec_count; Proměnná, která počítá, kolik vektorů bylo vloženo do matice. Při vložení 15. vektoru je proměnná vynulována a je vyvolána inference.

void initCircularQueue(); Inicializační funkce pro kruhovou frontu.

void enqueueCircular(); Funkce mění pozici indexu head v kruhovém bufferu.

void dequeueCircular(); Funkce mění pozici indexu tail v kruhovém bufferu.

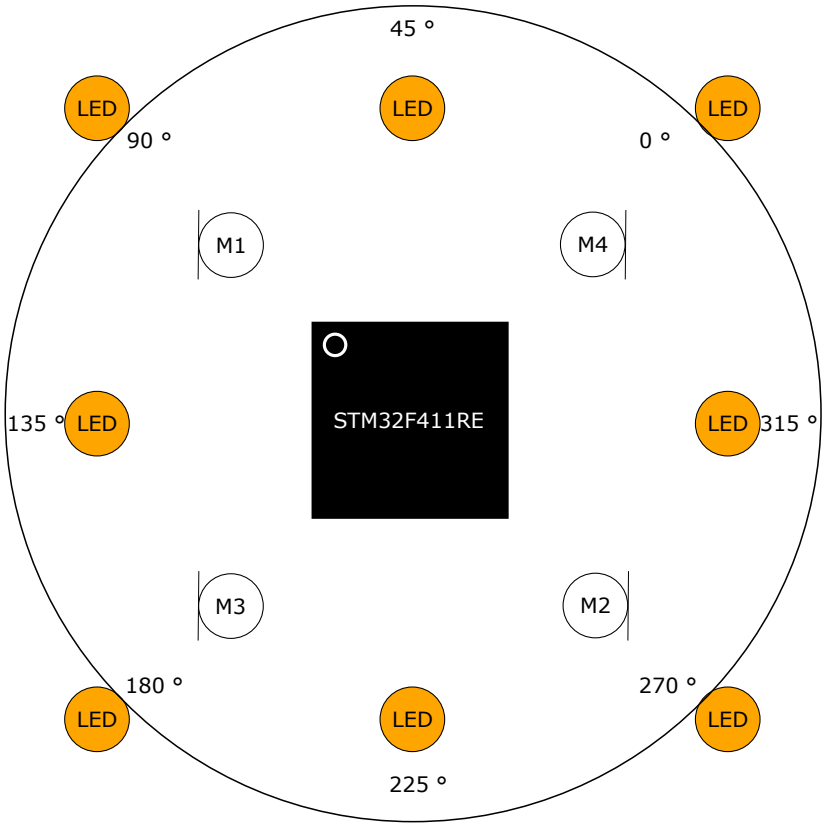
void addData(); Spojení funkce *enqueueCircular* a *dequeueCircular*. Používá se po prvním naplnění kruhové fronty.

void takeData(); Vyjme všechna data z kruhové fronty od indexu *head* do indexu *tail*.

Lokalizační část programu začíná také přerušením vyvolaným periferií I^{2}S. Výpočet odhadů se spustí, jakmile přerušení vyvolají obě periferie. Funkce pro výpočet odhadu vyžaduje rozdělení dat do menších kusů podle rovnice 5.2. Výsledkem je pole, jehož prvním elementem je odhadovaný úhel. Ten je při detekci sirény vypisován do konzole a zároveň určuje, která LED se při detekci rozsvítí. Na obrázku 5.14 je znázorněno rozložení mikrofonů s mikrokontrolérem, signalizačními LED a kružnicí s úhly.

$$N_{Chunk} = \frac{N_{Microphones} \cdot f_{sample\_rate}}{1000} \tag{5.2}$$

- N_{Chunk} je počet menších kusů
- $N_{Microphones}$ je počet použitých mikrofonů
- f_{sample_rate} je nastavená vzorkovací frekvence



Obrázek 5.14: Rozložení mikrofonů s mikrokontrolérem a signalizačními LED

```
void AudioProcess()
{
    int32_t estim_angle[2];
    int16_t audioLocChunkMic12[CHUNK_SIZE];
    int16_t audioLocChunkMic34[CHUNK_SIZE];
    for(int i=0; i<SAMPLES_2_PROCESS; i+=CHUNK_SIZE)
    {
        memcpy(audioLocChunkMic12,
                &audioLocInMic12[i],
                CHUNK_SIZE*sizeof(int16_t));
        memcpy(audioLocChunkMic34,
                &audioLocInMic34[i],
                CHUNK_SIZE*sizeof(int16_t));

        if(AcousticSL_Data_Input(
                (int16_t *)&audioLocChunkMic12[0],
                (int16_t *)&audioLocChunkMic12[1],
                (int16_t *)&audioLocChunkMic34[0],
                (int16_t *)&audioLocChunkMic34[1],
                &hacoust1) == 1U)
        {
            AcousticSL_Process(
                    (int32_t*) &estim_angle,
                    &hacoust1);

            estimated_angle_output = (int16_t)estim_angle[0];
        }
    }
}
```

Obrázek 5.15: Lokalizace zdroje zvuku pomocí mikrokontroléru

int32_t estim_angle[2]; Pole na indexu 0 obsahuje odhadovaný úhel směru zdroje zvuku.

int16_t audioLocChunkMicXY[CHUNK_SIZE]; Buffer slouží k rozdělení audio záznamu mikrofonů X a Y na menší díly. Rovnice 5.2 popisuje jeho výpočet a velikost.

int16_t audioLocInMicXY[256]; Buffer s původním 32ms audiozáznamem z mikrofonů X a Y.

uint32_t AcousticSL_Data_Input(); Knihovní funkce nastavené struktuře připojí vstupní data.

uint32_t AcousticSL_Proces(); Knihovní funkce odhadne úhel směru zdroje zvuku.

int16_t estimated_angle_output; Globální proměnná použita k uložení odhadovaného úhlu z pole estim_angle.

Výpočet MFCC a případnou inferenci obstarává funkce $mfcc\_circularBuf\_routine$. Obrázek 5.12 reprezentuje chování funkce. Její vývojový diagram je na obrázku 5.16. Funkce spočítá vektor s kepstrálními koeficienty a vloží ho do kruhové fronty. Pokud se jedná o patnáctý spočtený vektor, data z kruhové fronty jsou vyjmuta a předána neuronové síti, která vytvoří pravděpodobnostní odhady pro všechny třídy. Pokud je odhad sirény ze tří po sobě jdoucích inferencí větší než 85 %, siréna je programem detekována. Na konzoli se přes USART vypíše odhadnutý úhel zdroje zvuku a jedna z osmi LED se rozsvítí ve směru zdroje zvuku.

int16_t mfcc_input; Vstupní vektor s uloženými audio daty připravenými pro výpočet MFCC.

float32_t mfcc_output; Výstupní vektor, který představuje vypočítané kepstrální koeficienty.

void compute_mfcc(); Funkce, jejíž vstupní parametry jsou předešlé dva vektory. Funkce z vektoru $mfcc\_input$ dopočítá 16 kepstrálních koeficientů a uloží je do vektoru $mfcc\_output$.

ai_float in_data; Vstupní vektor pro model neuronové sítě. Vektor je reprezentací kepstrální matice a jedná se o přeskládaná data z kruhové fronty.

ai_float out_data; Výstupní vektor pro model neuronové sítě. Vektor obsahuje deset hodnot, které představují pravděpodobnostní odhady jednotlivých labelů.

void perform_inference(); Funkce, jejíž vstupní parametry jsou předešlé dva vektory. Funkce z vektoru *in_data* dopočítá pravděpodobnostní odhady pro jednotlivé labely a vloží je do vektoru *out_data*.

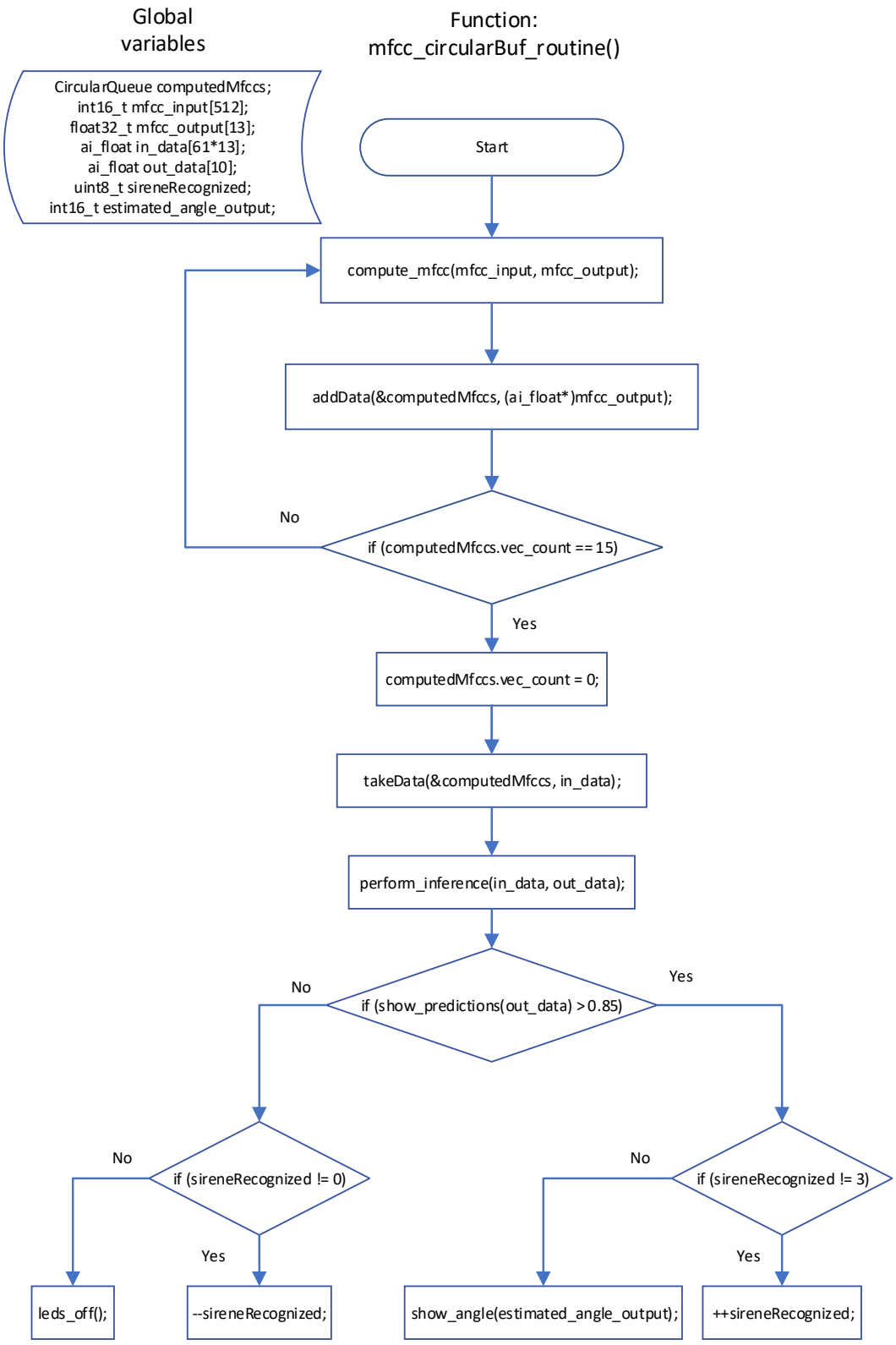
ai_float show_predictions(); Vstup funkce je vektor *out_data*. Funkce vypíše na konzoli přes USART všechny pravděpodobnostní odhady. Návratová hodnota funkce je rovna pravděpodobnostnímu odhadu sirény a je použita v podmínce.

uint8_t sireneRecognized(); Jedná se o stavové počítadlo, které je inkrementováno kdykoliv je siréna detekována do maximální hodnoty 3. Pokud siréna detekována není, počítadlo je dekrementováno do minimální hodnoty 0. Při dosažení hodnoty 3 je po další detekci sirény zobrazen směr zdroje zvuku pomocí LED a výpisu na konzoli. Všechny LED zhasínají v případě, kdy počítadlo dojde na hodnotu 0.

void leds_off(); Funkce, která zhasne všechny LED.

int16_t estimated_angle_output; Proměnná, ve které je uložen úhel odhadovaného směru zdroje zvuku vypočítaný knihovnou AcousticSL.

void show_angle(); Funkce vypíše na konzoli odhad úhlu, ve kterém je zdroj zvuku a rozsvítí LED v odhadovaném směru.

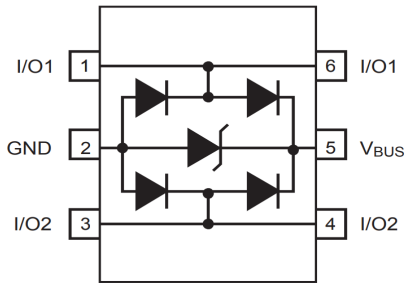


Obrázek 5.16: Vývojový diagram funkce mfcc_circularBuf_routine

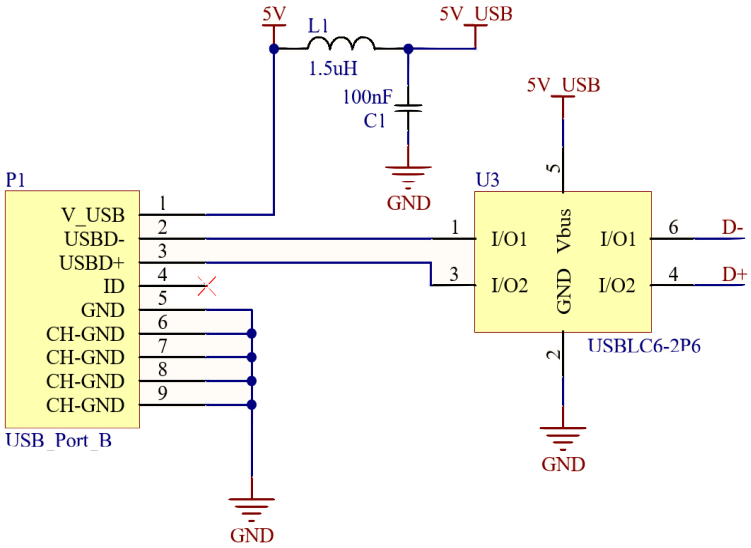
5.5 Návrh desky plošných spojů

Návrh je vytvořen v programu Altium Designer ve verzi 23.4.1.

Mini USB slouží k 5V napájení desky a komunikaci s počítačem. Vstup je chráněn ESD ochranou USBLC6-2P6.

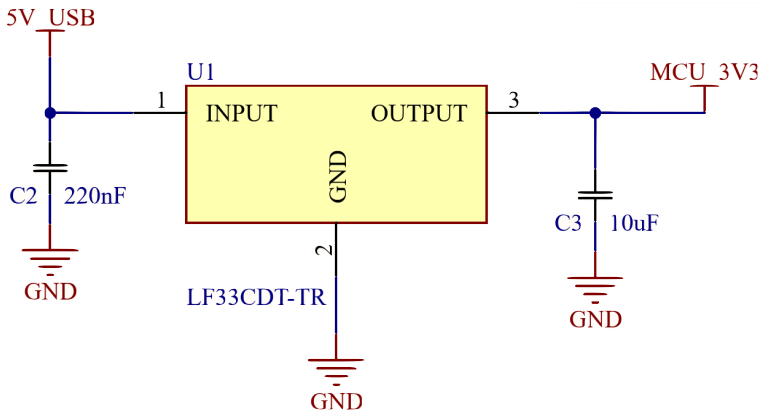


Obrázek 5.17: Diagram ESD ochrany USBLC6-2P6 [26]



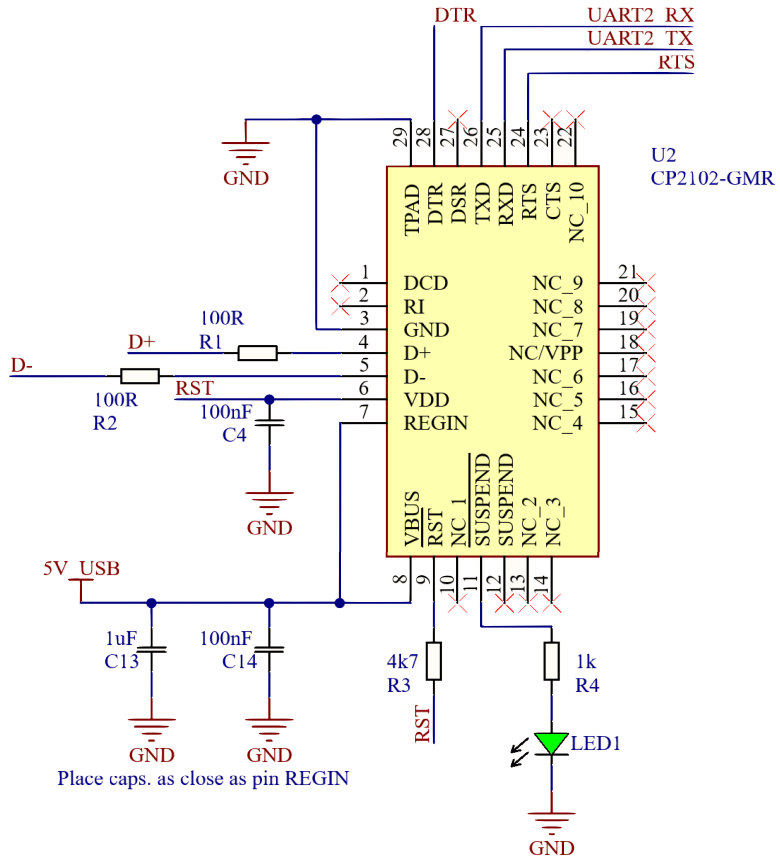
Obrázek 5.18: Schématické zapojení USB mini s ESD ochranou

LDO regulátor LF33CDT-TR slouží k snížení 5V napětí na 3,3 V, které požadují mikrofony i mikrokontrolér. Velikosti blokovacích kondenzátorů jsou převzaty z katalogové listu [27].



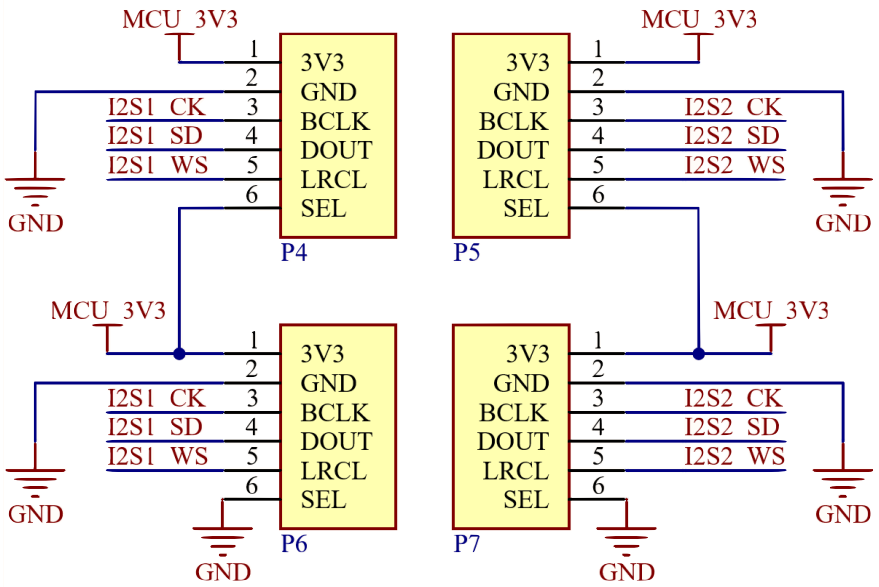
Obrázek 5.19: Schématické zapojení LDO LF33CDT-TR

USB na USART konverzi při komunikaci obstarává integrovaný obvod CP2102-GMR. Na pin $\overline{\text{SUSPEND}}$ je připojena signalizační LED, která svítí, když probíhá komunikace mezi PC a mikrokontrolérem. Zapojení integrovaného obvodu je převzato z katalogového listu [28].



Obrázek 5.20: Schématické zapojení CP2102-GMR

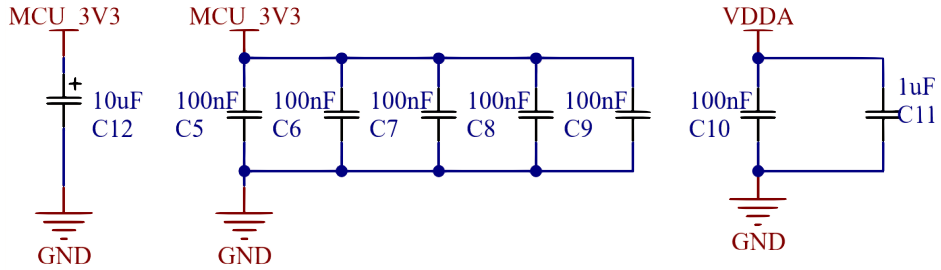
Mikrofony SPH0645 jsou součástí samostatného modulu a do desky plošných spojů jsou připojeny pomocí pinových headerů.



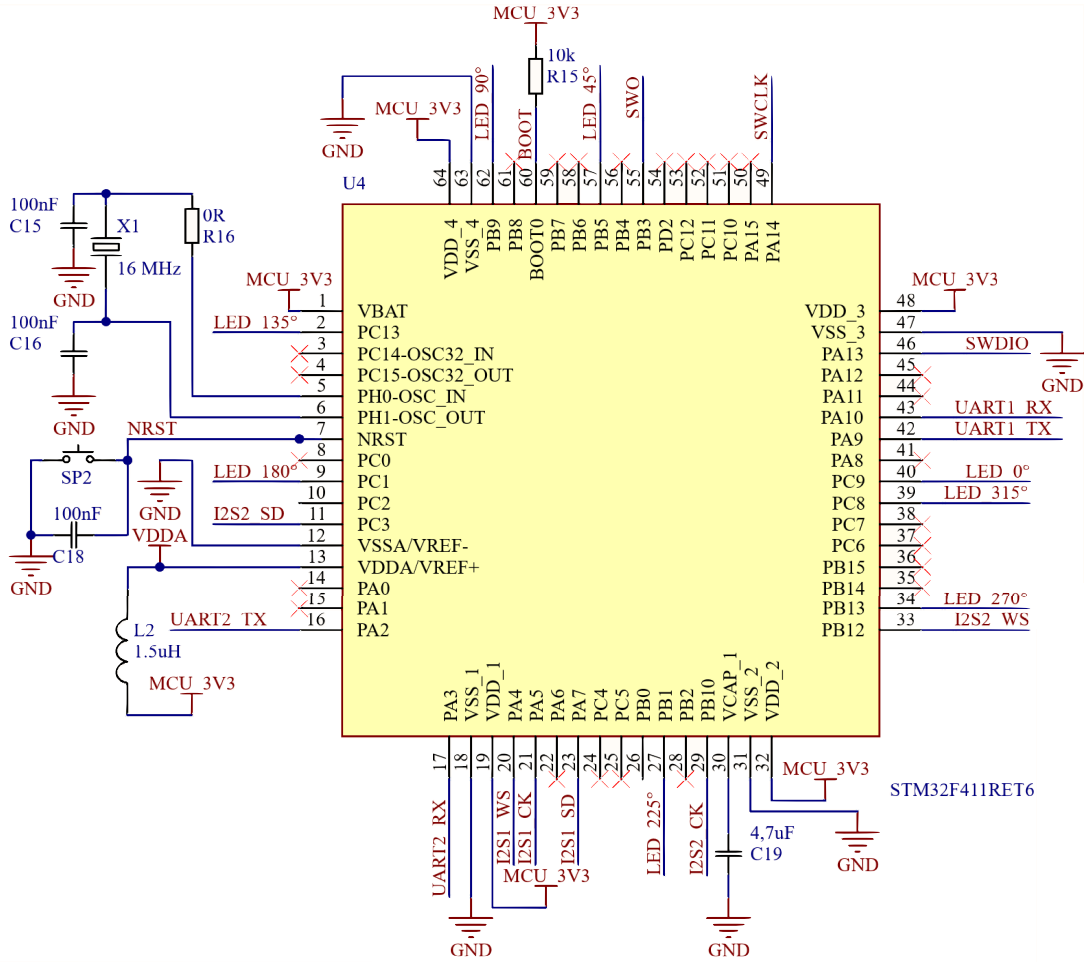
Obrázek 5.21: Pinové lišty pro mikrofoní moduly

Mikrokontrolér má 3,3V napájecí napětí. Ke každému napájecímu pinu VDD je připojen jeden externí keramický kondenzátor s kapacitou 100 nF tak, aby jejich vzdálenost byla minimální. K pinu 1 je připojen navíc tantalový 10μF kondenzátor. Pin VBAT je s externím 100nF keramickým kondenzátorem připojen k 3,3V napětí. Analogový napájecí pin je připojen k 100nF keramickému a 1μF tantalovému kondenzátoru. Analogové a digitální napájecí napětí je oddělené ferritovým korálkem o velikosti 1,5 μH. K pinu VCAP je připojen keramický 4,7μF kondenzátor [29]. Ostatní periferie jsou zapojeny podle návrhu z STM32CubeMX. K osmi pinům je připojeno osm signalizačních LED pro určení směru zdroje zvuku. Komunikace se stolním počítačem probíhá přes periferii USART2. Programování mikrokontroléru je možné za použití čtyř SWD pinů. K tomu je potřeba využít externí programátor.

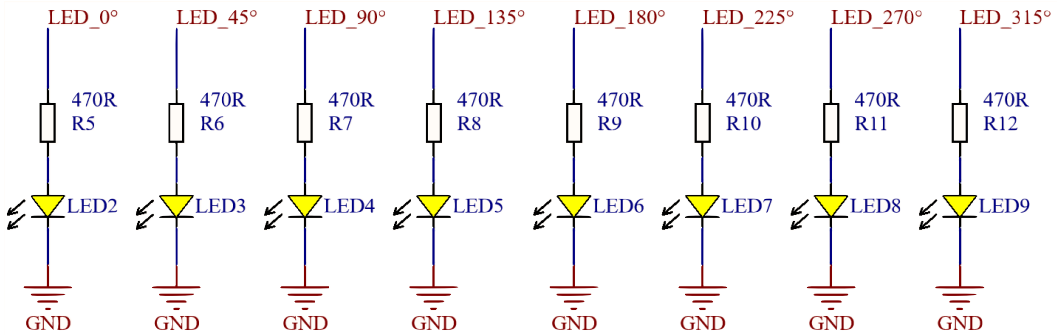
Na desce plošných spojů je připraven WiFi modul s mikrokontrolérem ESP-12E pro případnou bezdrátovou komunikaci s počítačem. Modul by měl zastávat funkci TCP/IP klienta a s mikrokontrolérem STM32F411 spojen přes periferii USART1.



Obrázek 5.22: Vyhlazovací kondenzátory mikrokontroléru STM32F411RE

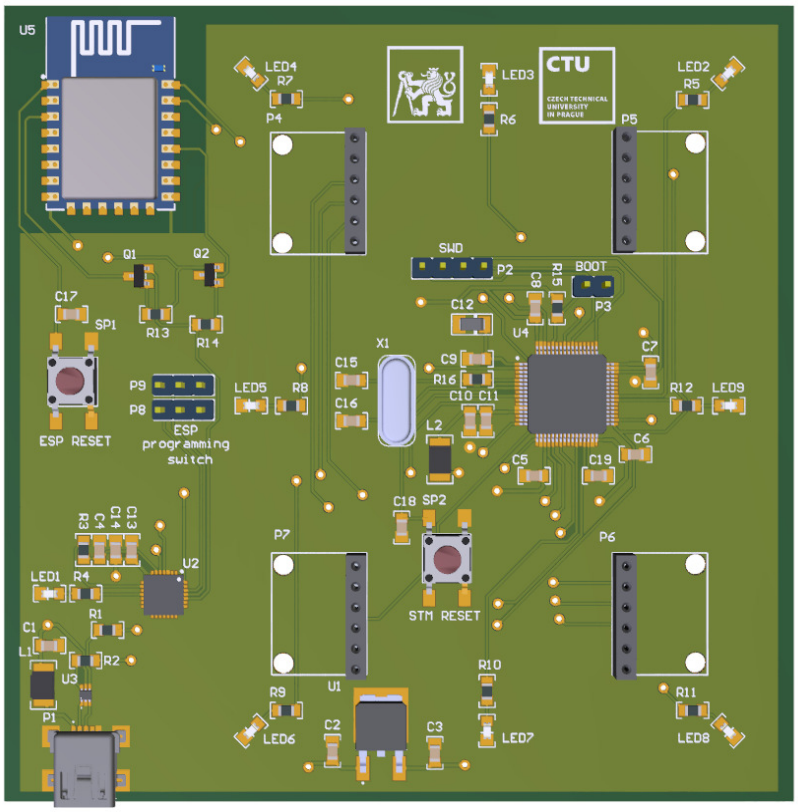


Obrázek 5.23: Schématické zapojení mikrokontroléru STM32F411RE



Obrázek 5.24: Schéma indikačních LED

Mikrofony jsou na desce plošných spojů umístěny do čtverce s 90mm diagonálami. Rozměry desky jsou 100x100 mm. Všechny součástky jsou umístěny v horní vrstvě. Signály jsou vedeny především horní a spodní vrstvou. V jedné vnitřní vrstvě je rozlit polygon s 5V a 3,3V napájecím napětím. V druhé vnitřní vrstvě a obou vnějších vrstvách se nachází rozlitá zem. Signály periferií I^2S mají maximální rozdíl zpoždění 400 ps. Na obrázku 5.25 je vyrenderován 3D návrh desky plošných spojů.

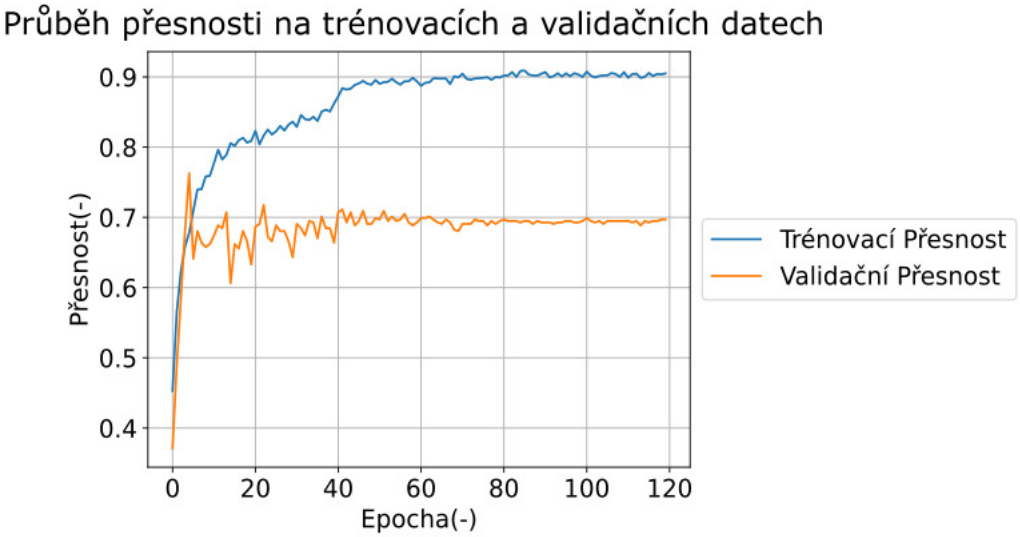


Obrázek 5.25: Vyrenderovaný 3D model desky plošných spojů

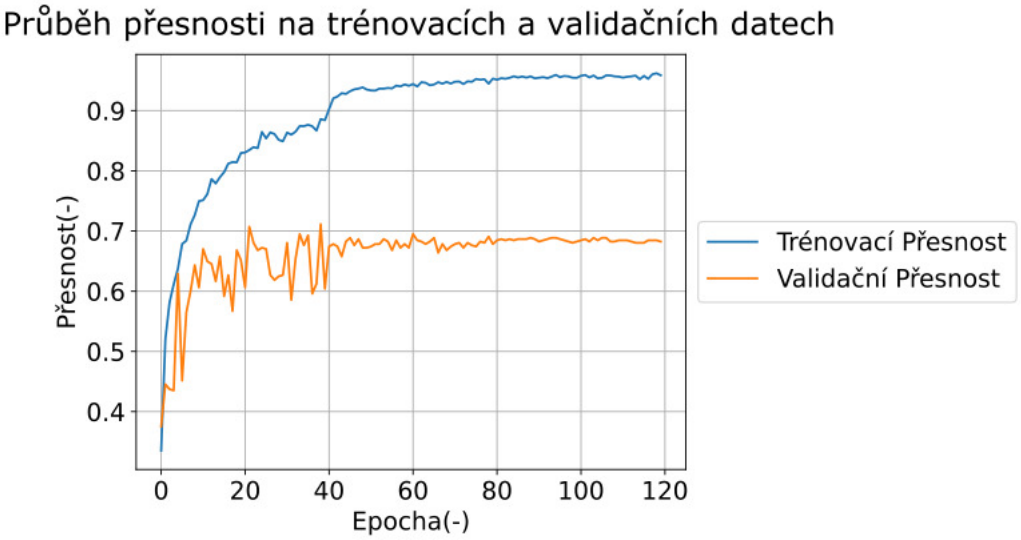
Kapitola 6
Výsledky

6.1 Přesnost a rychlost modelů

Modely Baseline i ESC dosahují při použití desetinásobné křížové validace 95% přesnost na trénovacích datech. Na validačních datech mají modely nejvyšší úspěšnost 69 %. Na obrázcích 6.1 a 6.2 jsou znázorněny průběhy přesností během trénování. Z obrázků lze usoudit, že jsou oba modely přetrénované.

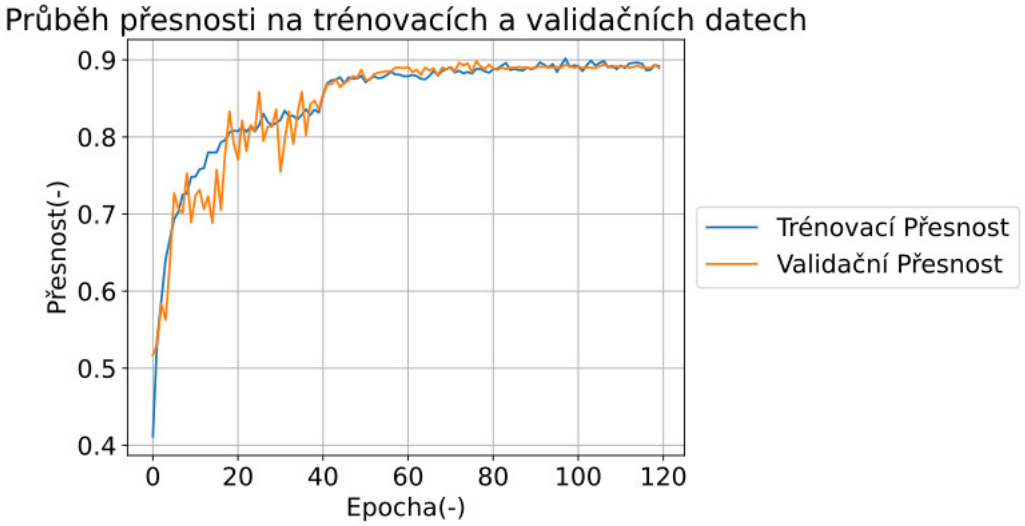


Obrázek 6.1: Přesnost modelu Baseline za použití desetinásobné křížové validace

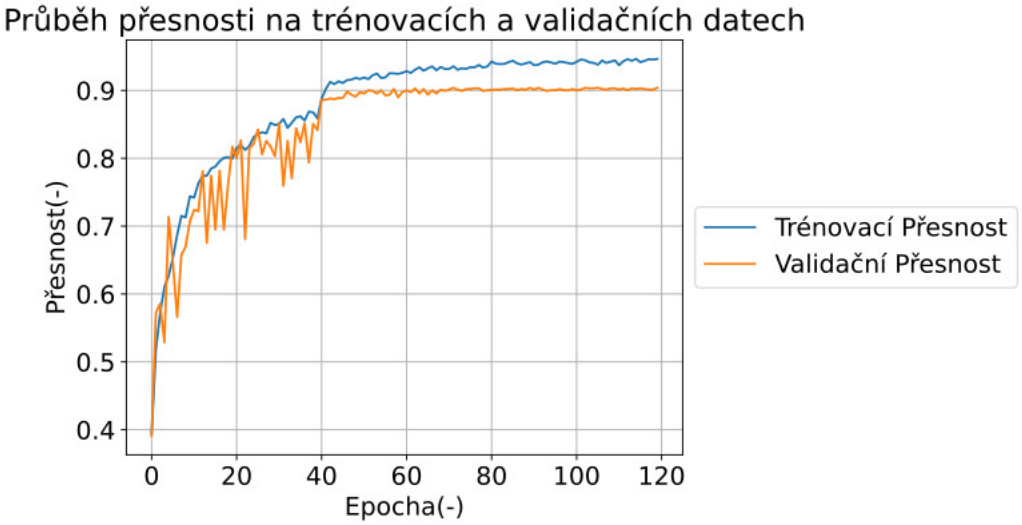


Obrázek 6.2: Přesnost modelu ESC za použití desetinásobné křížové validace

Náhodné rozdělení datasetu na testovací a validační skupinu v poměru 8:2 vedlo k vyšší přesnosti. Pro model Baseline je přesnost na trénovacích i validačních datech rovna 89 %. Model ESC dosahuje přesnosti 95 % na trénovacích datech a 90 % na datech testovacích. Na obrázcích 6.3 a 6.4 jsou znázorněny průběhy přesností během trénování.

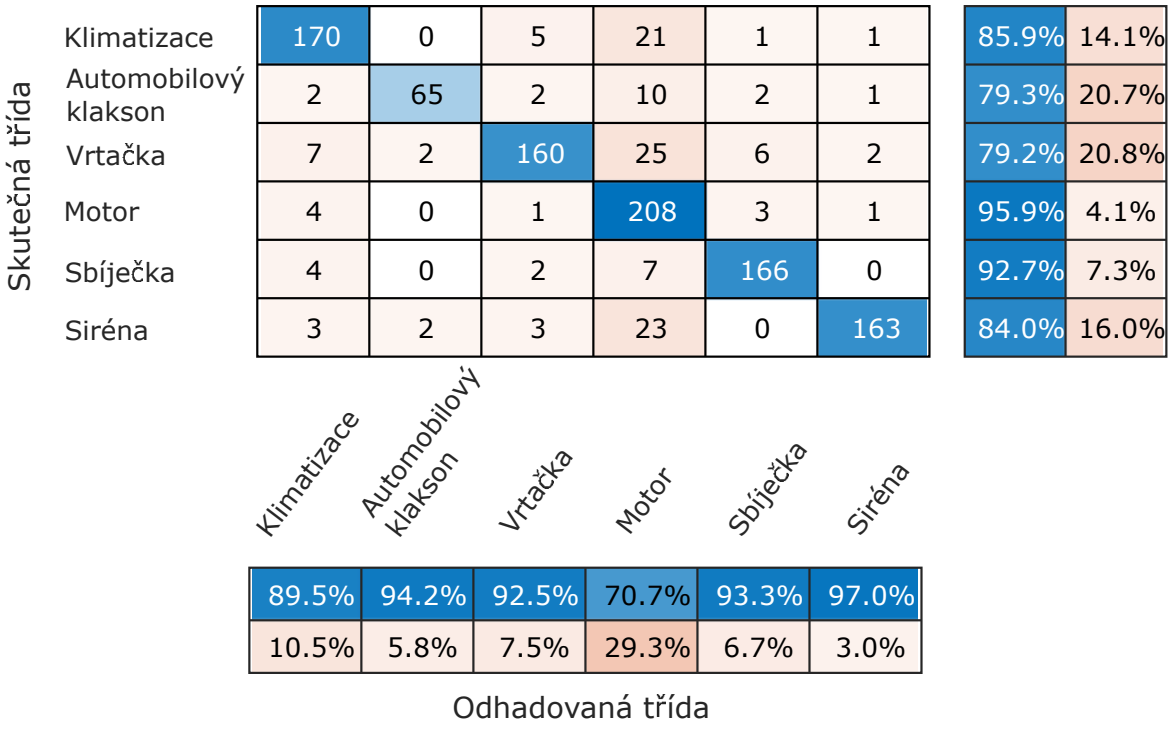


Obrázek 6.3: Přesnost modelu Baseline za použití náhodného rozdělení v poměru 8:2

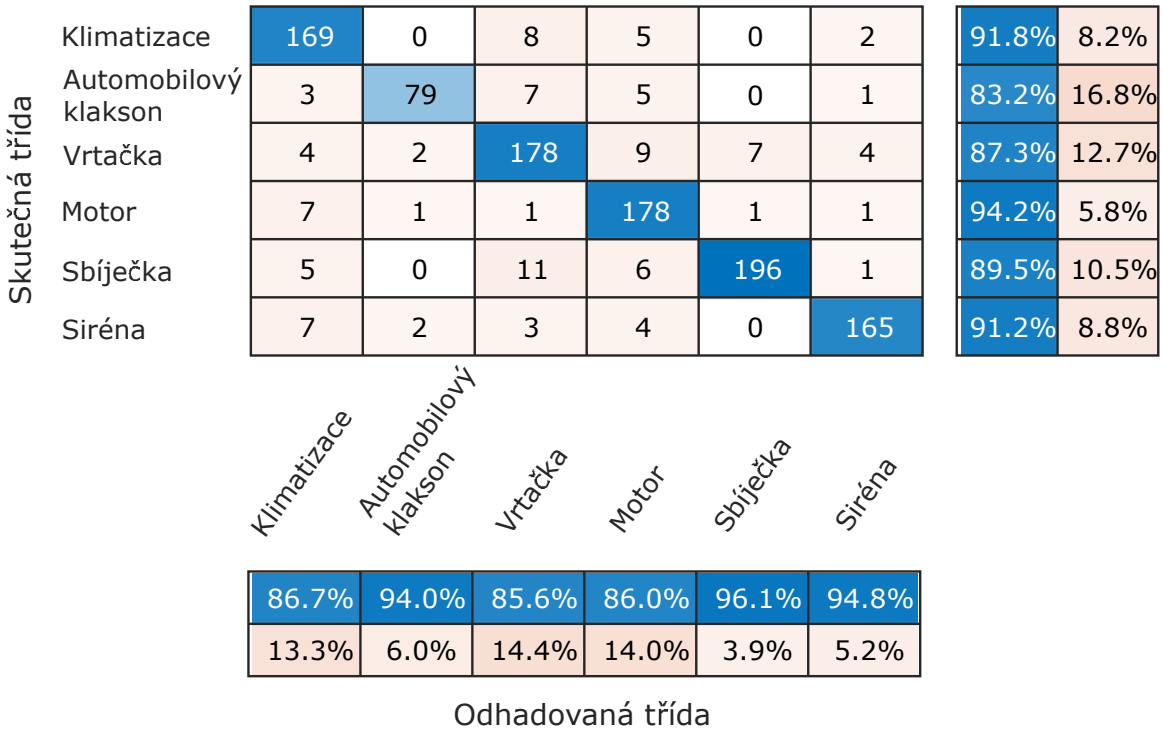


Obrázek 6.4: Přesnost modelu ESC za použití náhodného rozdělení v poměru 8:2

Matice záměn (*confussion matrices*) pro oba modely jsou zobrazeny na obrázcích 6.5, 6.6 a znázorňují výsledek na validačních datech. Na diagonále je absolutní počet správně klasifikovaných jednotlivých skupin. Mimo diagonálu jsou chybně určené odhady. Tabulka vpravo znázorňuje procentuální zastoupení, kolikrát byla třída správně určena. Tabulka pod maticí znázorňuje procentuální zastoupení, kolikrát předpověď odpovídala skutečné třídě.



Obrázek 6.5: Výsledná matice záměn po natrénování modelu Baseline



Obrázek 6.6: Výsledná matice záměn po natrénování modelu ESC

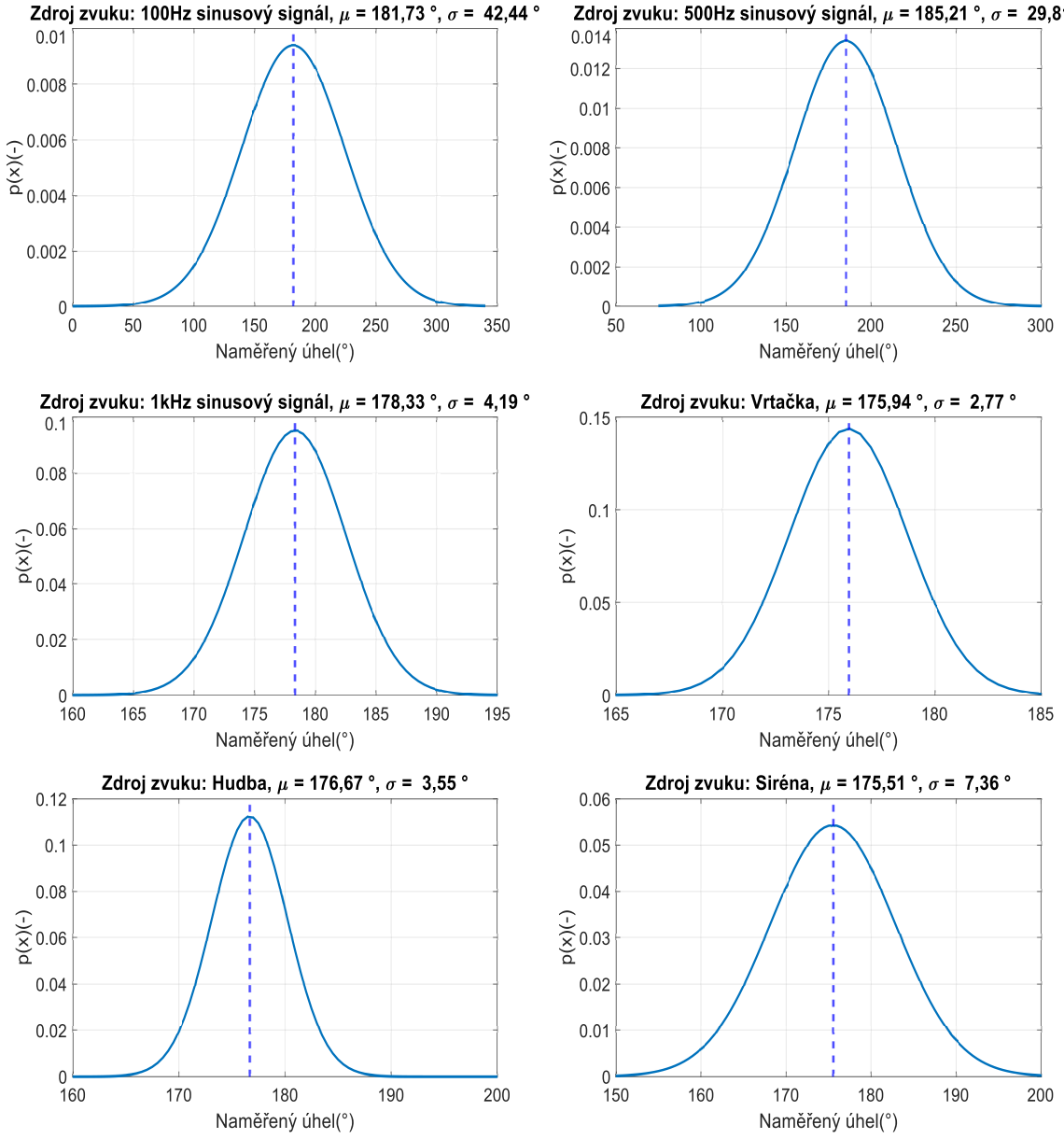
Doba trvání převodu na kepstrální koeficienty, lokalizace zdroje zvuku a inference je změřena pomocí vnitřního časovače mikrokontroléru TIM10. Paměťové požadavky jsou vypočteny během nahrávání programu do mikrokontroléru. Čas pro výpočet MFCC a lokalizace zdroje zvuku není závislý na složitosti modelu. Potřebné časy a paměťové požadavky jsou znázorněné v tabulce 6.1.

Tabulka 6.1: Časová a paměťová náročnost obou modelů

Parametr	Hodnota
Doba výpočtu MFCC	2,84 ms
Doba lokalizace zdroje zvuku	5,12 ms
Model Baseline	
Doba inference	9,81 ms
Využití paměti SRAM modelem	80,57 kB
Využití paměti FLASH modelem	108,43 kB
Celkové využití paměti SRAM programem	97,50 kB
Celkové využití paměti FLASH programem	310,64 kB
Model ESC	
Doba inference	36,08 ms
Využití paměti SRAM modelem	52,44 kB
Využití paměti FLASH modelem	108,43 kB
Celkové využití paměti SRAM programem	107,00 kB
Celkové využití paměti FLASH programem	307,50 kB

6.2 Přesnost lokalizace zvuku

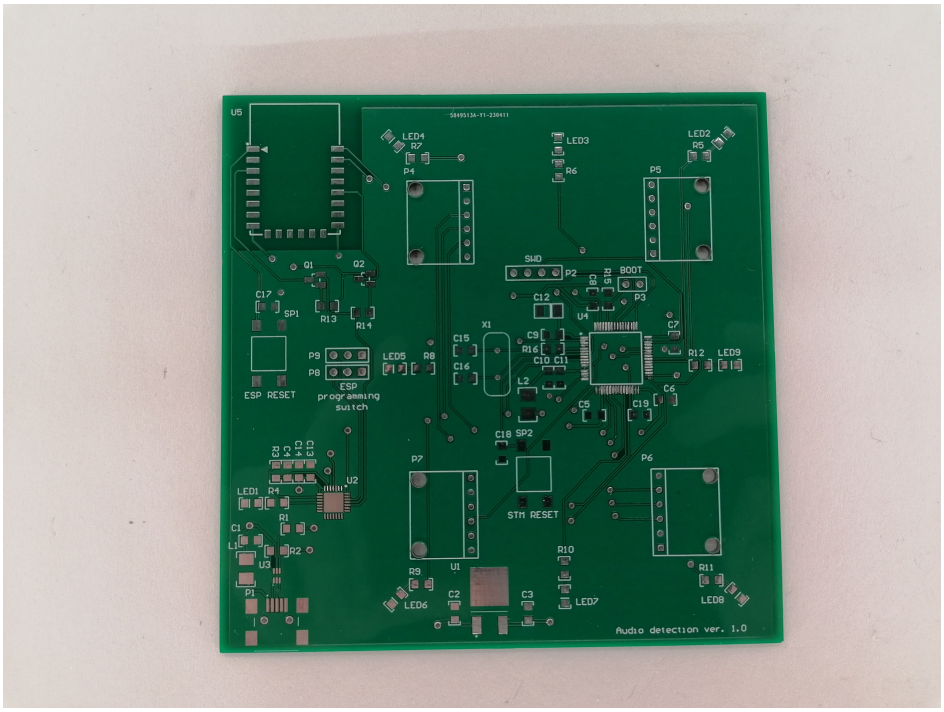
Rozlišení mikrofonního pole je měřeno přiložením bodového zdroje zvuku 30 cm od zařízení. Zdroj vysílá zvuk z místa, kde systém lokalizuje 180 °. Při měření je vysíláno šest různých zvuků. Jedná se o 1kHz, 500Hz, 100Hz sinusový signál, zvuk sirény, hudby a vrtačky. Rozdělení naměřených zdrojů zvuku jsou na obrázku 6.7.



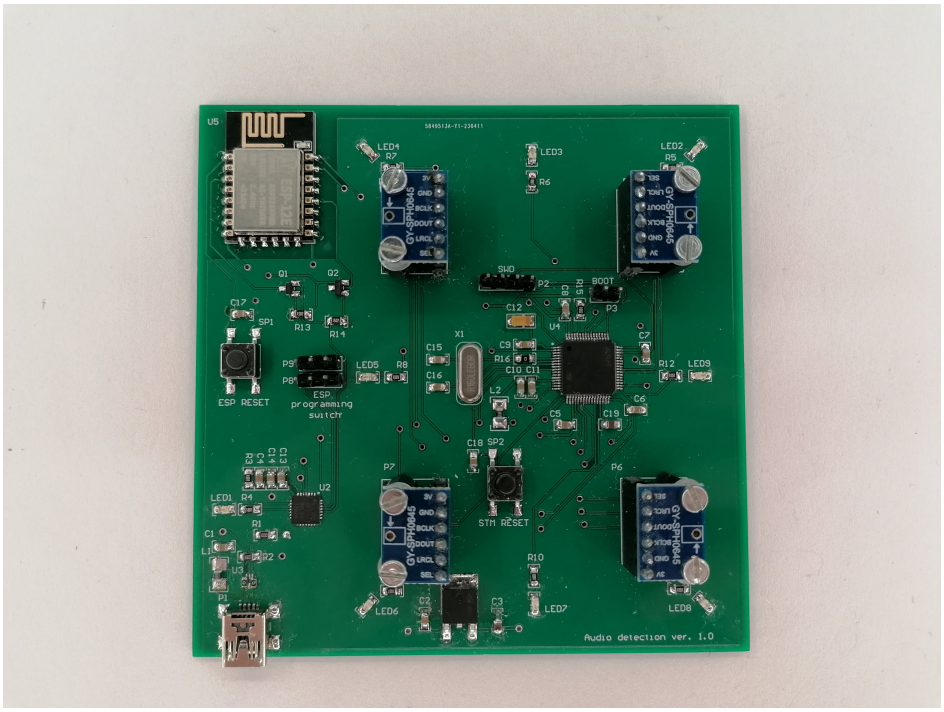
Obrázek 6.7: Normální rozdělení naměřených úhlů pro 6 různých zdrojů zvuku

6.3 Výsledná deska plošných spojů

Deska plošných spojů je vyrobena firmou JLCPCB. Součástky jsou ručně připájeny. Na obrázcích 6.8 a 6.9 je vidět výsledná deska před a po osazení. V tabulce 6.2 jsou vypsány všechny potřebné komponenty.



Obrázek 6.8: Deska plošných spojů před osazením



Obrázek 6.9: Deska plošných spojů po osazení

Tabulka 6.2: Seznam použitých součástek na desce plošných spojů

Součástka	Hodnota	Počet	Typ pouzdra
Keramický kondenzátor	100 nF	13	0805
Keramický kondenzátor	220 nF	1	0805
Keramický kondenzátor	1 μF	2	0805
Keramický kondenzátor	4,7 μF	1	0805
Keramický kondenzátor	10 μF	1	0805
Tantalový kondenzátor	10 μF	1	1206
Ferritový korálek	1,5 μH	2	1206
Rezistor	100 Ω	2	0805
Rezistor	470 Ω	8	0805
Rezistor	1 kΩ	4	0805
Rezistor	15 kΩ	2	0805
Krystalový oscilátor	16 MHz	1	0805
Pinový header	1x4	1	THT
Pinový header	1x3	2	THT
Pinový header	1x2	1	THT
Dutinkový pinový header	1x6	4	THT
LED zelená		1	0805
LED žlutá		8	0805
Mini USB port		1	THT
NMOS BSS138		2	SOT-23-3
Tlačítko		2	4pinové SMD
Napěťový regulátor LF33CDT-TR		1	TO-252-3
Převodník USB na USART CP2102-GMR		1	28pinové QFN
ESD ochrana USBLC6-2P6		1	SOT-666
Mikrokontrolér STM32F411RET6		1	LQFP64
WiFi modul ESP_12_E		1	

6.4 Konzolový výstup

Mikrokontrolér zasílá data pomocí periferie USART2. Na obrázku 6.10 je zachycen výstup pomocí konzolového programu PuTTY. Na začátku programu je zaslána zpráva o názvu modelu a jeho dimenzi vstupu a výstupu. Zbylé zprávy jsou pravděpodobnostní odhady jednotlivých tříd. Pokud je pravděpodobnostní odhad sirény nad 85 %, mikrokontrolér zašle informaci o lokaci sirény.

```
Model name      : network

Model signature : a8b0e807db557afaa718102843014e61

input[0] : (61, 16, 1)

output[0] : (1, 1, 6)

Inference output..

Air conditioner: 0.000004

Car horn: 0.000449

Drilling: 0.000014

Engine idling: 0.014576

Jackhammer: 0.000000

Siren: 0.984957

Siren detected at 250 °
```

Obrázek 6.10: Terminálový výstup mikrokontroléru

Kapitola 7
Diskuze

7.1 Srovnání modelů

Při použití desetinásobné křížové validace dosahují modely Baseline i ESC 95% úspěšnosti na trénovacích datech. Na validačních datech mají modely výrazně nižší přesnost. Můžeme předpokládat, že jsou oba modely přeučeny. U náhodného rozdělení datsetu v poměru 8:2 model Baseline dosahuje vysoké přesnosti 89 % na trénovacích i validačních datech. Tato přesnost naznačuje schopnost modelu efektivně rozpoznávat jednotlivé třídy na nových neznámých datech. Model ESC dosahuje přesnosti 95 % na trénovacích datech. Na validačních datech dosáhl mírně nižší přesnosti 90 %. Odchylka může naznačovat přítomnost mírného přetrénování modelu na trénovacích datech. Rozšíření a vyvážení trénovacího datasetu může dopomoci k zmenšení odchylky.

Přetrénování modelu při desetinásobné validaci je neobvyklé. Je to pravděpodobně způsobeno tím, že jednotlivé třídy jsou v složkách rozděleny chybným způsobem a každá složka obsahuje specifickou vlastnost tříd.

7.2 Doba zpracování signálu

Klíčovým faktorem je časové omezení vyvolané přerušením každých 16 ms pomocí periferie I^2S. Omezení vyžaduje, aby byl výpočet MFCC a GCC-PHAT dokončen během tohoto časového intervalu. Časová náročnost vyplývá z potřeby nepřetržitého zpracování zvukových dat v reálném čase. Součet časových nároků pro výpočet MFCC a GCC-PHAT je 7,96 ms. Důležitým aspektem je také rychlost vyvolaných inferencí, které jsou provedeny každých 15 přerušení. Inferenční výpočet je nutné dokončit do 120 ms. Modely Baseline i ESC tuto podmínku splňují s velikou časovou rezervou. Důvod, proč nejsou použity výpočetně náročnější modely, je paměťové omezení mikrokontroléru STM32F411RE. Použitím mikrokontroléru STM32F469VI (180 MHz, 2 MB flash, 384 kB RAM) by bylo možné nahrát složitější a časově náročnější model.

7.3 Zhodnocení výsledků

Implementovaný model na mikrokontroléru nedosahuje takové přesnost v klasifikaci všech tříd, jako tomu bylo na validačních datech. Šum byl ve většině případů mikrokontrolérem vyhodnocen jako sbíječka. Mikrokontrolér třídy sbíječka a vrtačka v některých případech zaměňuje. Při implementování modelu, který byl natrénován na všech datech

z datasetu UrbanSound8k, nebylo možné rozpoznat sirénu od pouliční hudby. V navazující práci by bylo vhodné použít výkonnější mikrokontrolér a implementovat složitější model. Případně by bylo vhodné rozšířit dataset o upravená data, případně vytvořit vlastní data, která budou obsahovat nahrávky přímo z daného mikrofonu.

Výsledky měření lokalizace zvuku ukázaly odchylky použitého algoritmu v kombinaci s mikrofony SPH0645lm4h. Zdroje zvuku 100Hz a 500Hz sinusového signálu prokazovaly znepokojující směrodatné odchylky. Při lokalizaci systém vrátí nejistý výsledek. Ostatní změřené zdroje zvuku dosahovaly směrodatných odchylek pod 10 °. Takový výsledek je uspokojivý pro přibližné lokalizování zdroje. Pro budoucí práci by bylo vhodné implementovat nebo vytvořit novou knihovnu přímo určenou k lokalizaci zvuku. Knihovna AcousticSL není dobře zdokumentována, vysvětlení některých parametrů v dokumentaci úplně chybí.

7.4 Doporučená vylepšení

První verze desky plošných spojů obsahuje drobné chyby. Chybí montážní otvory pro distanční sloupky, které jsou nyní nahrazeny samolepicími podložkami. V první verzi je ESD ochrana realizována součástkou USBLC6-2P6 s pouzdrem SOT-666. Pro ruční pájení je vhodnější použít pouzdro SOT23-6L, které je o poznání větší. Nedostatkem je i absence napájecího a zemnícího pinu u programovacího rozhraní SWD. Při programování je potřeba mít připojenou desku k napájecímu zdroji přes mini USB a zároveň mít připojený programátor k rozhraní SWD. Aby programování správně fungovalo, je potřeba propojit země napájecího zdroje a programátoru. Poslední drobností je absence stínění na desce pomocí prokovů skrz zemnící plochy. Vylepšením diplomové práce by bylo zprovoznění modulu ESP-12E na desce a vytvořením grafického uživatelského rozhraní na straně počítače, které by interagovalo s přijatými zprávami.

Doporučení pro navazující práce by bylo implementovat i jiné metody strojového učení jako například SVM (*Support Vector Machine*) nebo metodu K nejbližších sousedů a porovnat výsledky s modelem konvoluční neuronové sítě.

Kapitola 8
Závěr

Tato diplomová práce měla za cíl implementovat metodu strojového učení pro klasifikaci zdroje zvuku a určování směru, ze kterého zvuk přichází. Práce se zaměřila na klasifikaci pomocí konvoluční neuronové sítě a lokalizaci zdroje zvuku pomocí zobecněné křížové korelace s fázovou transformací. Pro zpracování zvuku byla vybrána technika Melových frekvenčních kepstrálních koeficientů, která poskytuje vhodnou reprezentaci zvuku pro trénování neuronové sítě. Pomocí jazyka Python byly natrénovány a validovány dva modely s různou architekturou, které byly následně úspěšně nahrány do mikrokontroléru STM32F411RE.

V rámci práce byla navržena a realizována deska plošných spojů obsahující mikrofony, mikrokontrolér, osm LED pro signalizaci směru sirény a modul ESP-12E, který lze v budoucnu použít pro bezdrátový přenos dat. Na desce bylo v rámci experimentů změřeno rozlišení, které mikrofony společně s metodou zobecněné křížové korelace s fázovou transformací dokázaly poskytnout. Pomocí vnitřního časovače byly změřeny doby potřebné k výpočtu kepstrálních koeficientů, provedení křížové korelace a inference. V rámci implementace FW byly zaznamenány nároky obou modelů na paměti SRAM a flash.

Seznam použité literatury

[1] Dutta, Lachit a Bharali, Swapna. "Tinyml meets iot: A comprehensive survey". In: *Internet of Things* 16 (2021), s. 100461.

[2] Warden, Pete a Situnayake, Daniel. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[3] Marcus, Aaron. *Design, User Experience, and Usability: Users and Interactions: 4th International Conference, DUXU 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II*. Springer, 2015.

[4] Hurwitz, Judith a Kirsch, Daniel. "Machine learning for dummies". In: *IBM Limited Edition* (2018).

[5] Metz, Charles E. "Basic principles of ROC analysis". In: *Seminars in nuclear medicine*. Sv. 8. 4. Elsevier. 1978, s. 283–298.

[6] Zhang, Aston et al. "Dive into deep learning". In: *arXiv preprint arXiv:2106.11342* (2021).

[7] Hastie, Trevor et al. *The elements of statistical learning: data mining, inference, and prediction Second Edition*. Springer, 2017.

[8] Rosenblatt, Frank. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[9] *Multi Layer Perceptron*. [online]. Dub. 2021,[cit. 20.5.2023]. Dostupné z: https://commons.wikimedia.org/wiki/File:MultiLayerPerceptron.png.

[10] Kite, Thomas. "Understanding PDM digital audio". In: *Audio precision* (2012).

[11] *Pulse density modulation*. [online]. Říj. 2007,[cit. 20.5.2023]. Dostupné z: https://commons.wikimedia.org/wiki/File:Pulse_density_modulation.svg.

[12] *Pulse code modulation*. [online]. Břez. 2014,[cit. 20.5.2023].Dostupné z: https://upload.wikimedia.org/wikipedia/commons/b/bf/Pcm.svg.

[13] Huang, Xuedong et al. *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice hall PTR, 2001.

[14] Knapp, Charles a Carter, Glifford. "The generalized correlation method for estimation of time delay". In: *IEEE transactions on acoustics, speech, and signal processing* 24.4 (1976), s. 320–327.

[15] STMicroelectronics. *STM32F4 Series*. [online]. [Cit. 21.5.2023]. Dostupné z: https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html.

[16] Knowles [online katalogový list]. *SPH0645LM4H-B Datasheet*. Rev. C, ún. 2017 [cit. 21.4.2023]. Dostupné z: https://www.knowles.com/docs/default-source/model-downloads/sph0645lm4h-b-datasheet-rev-c.pdf?Status=Master&sfvrsn=c1bf77b1_4.

[17] STMicroelectronics [online katalogový list]. *STM32F411xx Datasheet*. DS10314. Rev. 7, pros. 2017 [cit. 21.4.2023]. Dostupné z: https://www.st.com/resource/en/datasheet/stm32f411ce.pdf.

[18] Salamon, Justin, Jacoby, Christopher a Bello, Juan Pablo. "A dataset and taxonomy for urban sound research". In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, s. 1041–1044.

[19] ARM. *CMSIS-DSP Repository*. [online]. [Cit. 22.4.2023]. Dostupné z: https://github.com/ARM-software/CMSIS-DSP.

[20] STMicroelectronics [online uživatelský manuál]. *Getting started with X-CUBE-AI Expansion Package for Artificial Intelligence (AI)*. UM2526. Rev. 8, led. 2022 [cit. 22.4.2023]. Dostupné z: https://www.st.com/resource/en/user_manual/dm00570145-getting-started-with-xcubeai-expansion-package-for-artificial-intelligence-ai-stmicroelectronics.pdf.

[21] STMicroelectronics [online uživatelský manuál]. *Getting started with AcousticSL real-time sound source localization middleware*. UM2212. Rev. 2, říj. 2021 [cit. 22.4.2023]. Dostupné z: https://www.st.com/resource/en/user_manual/um2212-getting-started-with-acousticsl-realtime-sound-source-localization-middleware-stmicroelectronics.pdf.

[22] ARM. *CMSIS - Cortex Microcontroller Software Interface Standard*. [online]. [Cit. 22.4.2023]. Dostupné z: https://developer.arm.com/tools-and-software/embedded/cmsis.

[23] STMicroelectronics [online referenční manuál]. *STM32F411xx Reference manual*. RM0383. Rev. 3, lis. 2018 [cit. 25.4.2023]. Dostupné z: https://www.st.com/resource/en/user_manual/um2212-getting-started-with-acousticsl-realtime-sound-source-localization-middleware-stmicroelectronics.pdf.

[24] STMicroelectronics [online uživatelský manuál]. *STM32CubeMX for STM32 configuration and initialization C code generation*. UM1718. Rev. 40, ún. 2023 [cit. 26.4.2023]. Dostupné z: https://www.st.com/resource/en/user_manual/um1718-stm32cubemx-for-stm32-configuration-and-initialization-c-code-generation-stmicroelectronics.pdf.

[25] STMicroelectronics [online programovací manuál]. *STM32 Cortex®-M4 MCUs and MPUs programming manual*. PM0214. Rev. 10, březen. 2020 [cit. 26.4.2023]. Dostupné z: https://www.st.com/content/ccc/resource/technical/document/programming_manual/6c/3a/cb/e7/e4/ea/44/9b/DM00046982.pdf/files/DM00046982.pdf/jcr:content/translations/en.DM00046982.pdf.

[26] STMicroelectronics [online katalogový list]. *Very low capacitance ESD protection*. DS4164. Rev. 7, lis. 2015 [cit. 28.4.2023]. Dostupné z: https://www.st.com/resource/en/datasheet/usblc6-4.pdf.

[27] STMicroelectronics [online katalogový list]. *Very low drop voltage regulator with inhibit function*. DS0519. Rev. 7, květ. 2017 [cit. 28.4.2023]. Dostupné z: https://www.st.com/resource/en/datasheet/lfxx.pdf.

[28] Silicon labs [online katalogový list]. *SINGLE-CHIP USB-TO-UART BRIDGE*. Rev. 1.8, led. 2017 [cit. 28.4.2023]. Dostupné z: https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf.

[29] STMicroelectronics [online aplikační manuál]. *Getting started with STM32F4xxxx MCU hardware development*. AN4488. Rev. 7, říj. 2018 [cit. 28.4.2023]. Dostupné z: https://www.st.com/content/ccc/resource/technical/document/application_note/76/f9/c8/10/8a/33/4b/f0/DM00115714.pdf/files/DM00115714.pdf/jcr:content/translations/en.DM00115714.pdf.

Příloha A
Model Baseline ve frameworku Keras

```python
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Input, BatchNormalization

def DS_CNN_small(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    # Block 1
    x = Conv2D(64, (3, 3), padding='same', name='conv1')(inputs)
    x = BatchNormalization(name='bn1')(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same', name='pool1')(x)
    x = Dropout(0.25)(x)

    # Block 2
    x = Conv2D(32, (3, 3), padding='same', name='conv2')(x)
    x = BatchNormalization(name='bn2')(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same', name='pool2')(x)
    x = Dropout(0.25)(x)

    # Block 3
    x = Conv2D(16, (3, 3), padding='same', name='conv3')(x)
    x = BatchNormalization(name='bn3')(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same', name='pool3')(x)
    x = Dropout(0.4)(x)

    # Flatten and classify
    x = Flatten()(x)
    x = Dense(num_classes, activation='softmax', name='fc')(x)

    # Create model
    model = Model(inputs=inputs, outputs=x, name='DS_CNN_small')
    return model
model = DS_CNN_small((61,16,1), 6)
```

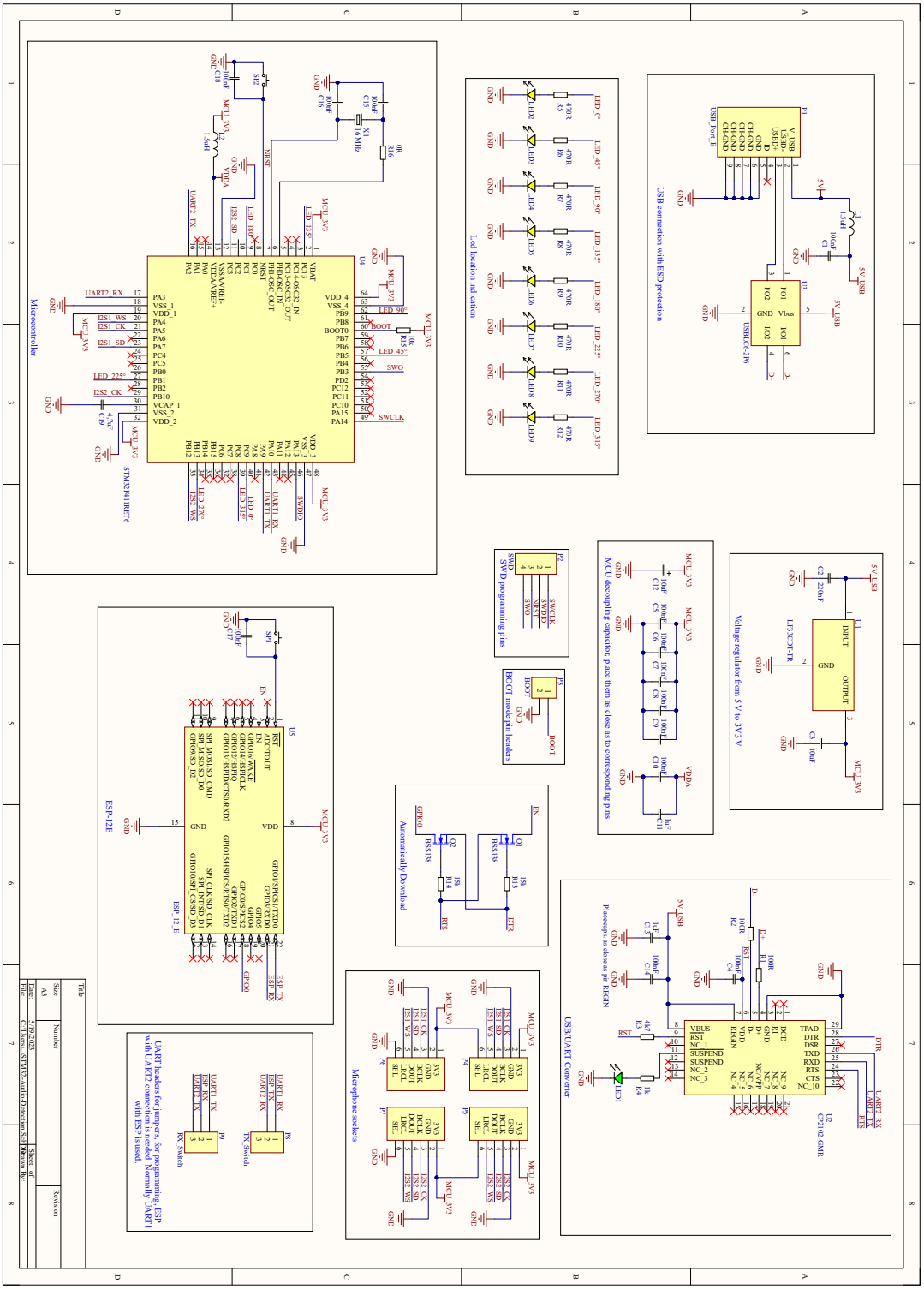
Obrázek A.1: Architektura modelu Baseline ve frameworku Keras

Příloha B
Model ESC ve frameworku Keras

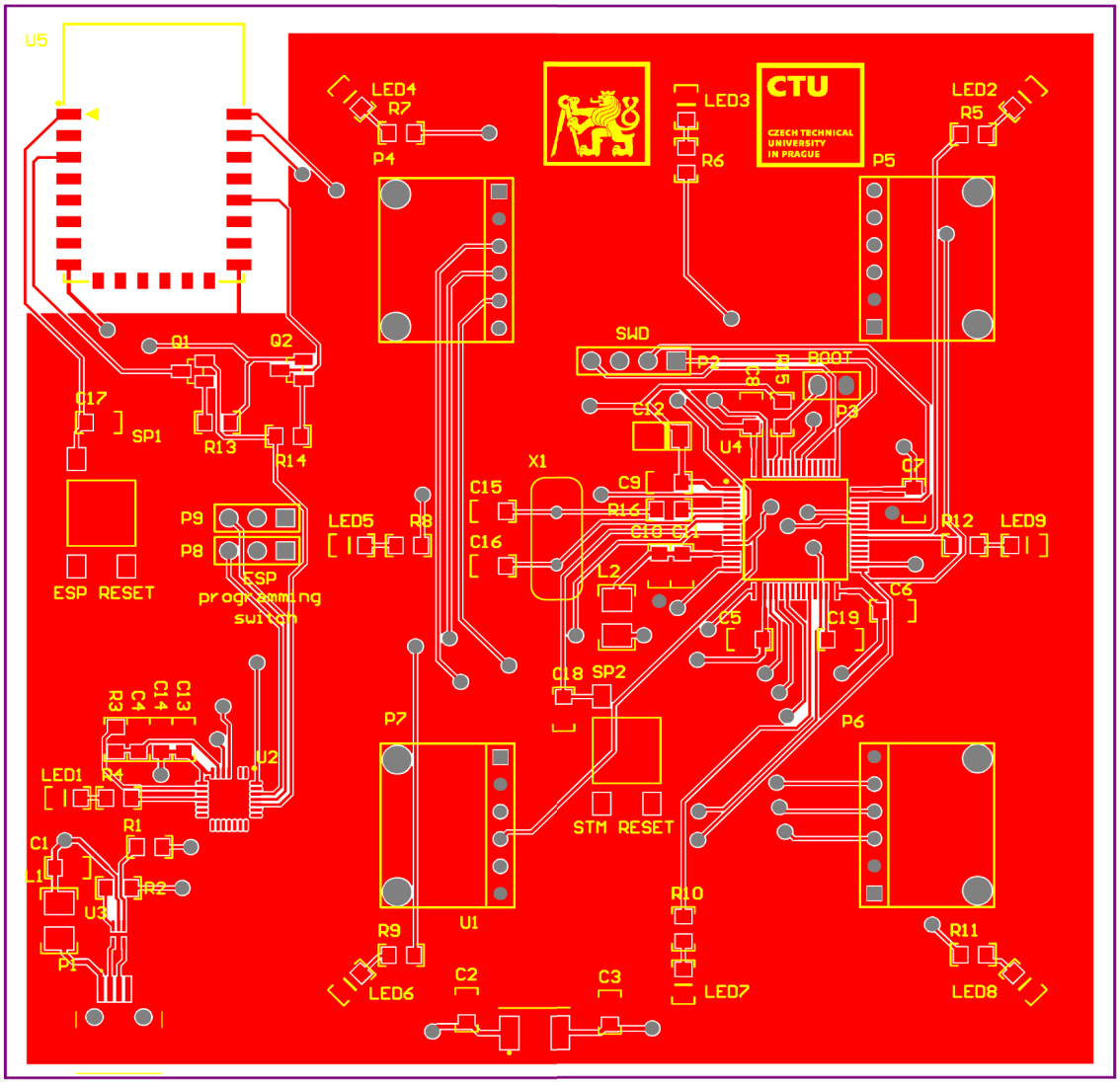
```python
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Input, BatchNormalization

def DS_CNN_small(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    # Block 1
    x = Conv2D(12, (5, 5), padding='same', name='conv1')(inputs)
    x = BatchNormalization(name='bn1')(x)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same', name='pool1')(x)
    x = Activation('relu')(x)

    # Block 2
    x = Conv2D(24, (5, 5), padding='same', name='conv2')(x)
    x = BatchNormalization(name='bn2')(x)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same', name='pool2')(x)
    x = Activation('relu')(x)

    # Block 3
    x = Conv2D(24, (5, 5), padding='same', name='conv3')(x)
    x = BatchNormalization(name='bn3')(x)
    x = Activation('relu')(x)

    # Flatten and classify
    x = Flatten()(x)
    x = Dropout(0.5)(x)
    x = Activation('relu')(x)

    x = Dropout(0.5)(x)
    x = Dense(num_classes, activation='softmax', name='fc')(x)

    # Create model
    model = Model(inputs=inputs, outputs=x, name='DS_CNN_small')
    return model
model = DS_CNN_small((61,16,1), 6)
```

Obrázek B.1: Architektura modelu ESC ve frameworku Keras

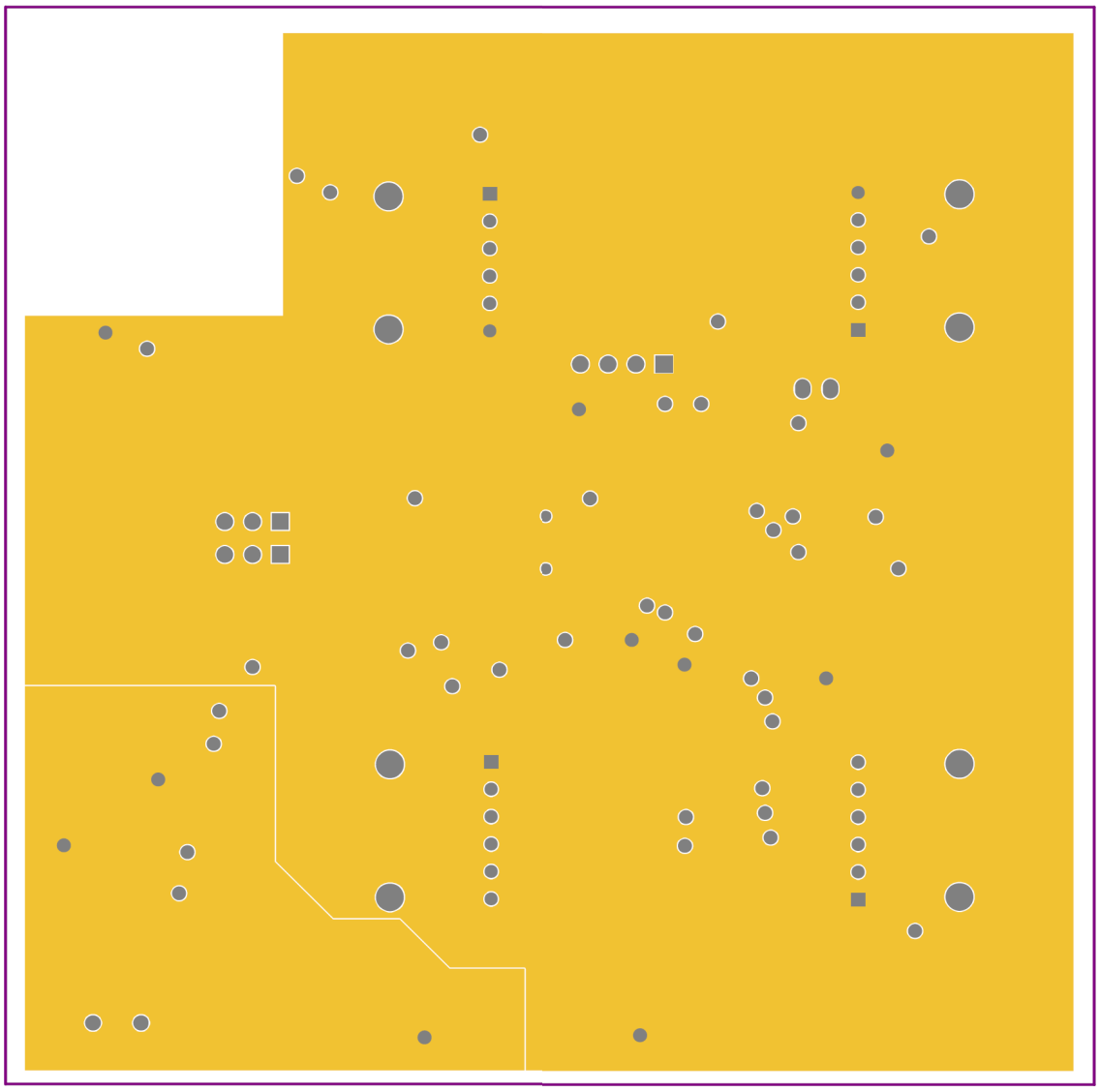
Příloha C
Elektrické schéma a jednotlivé vrstvy desky plošných spojů



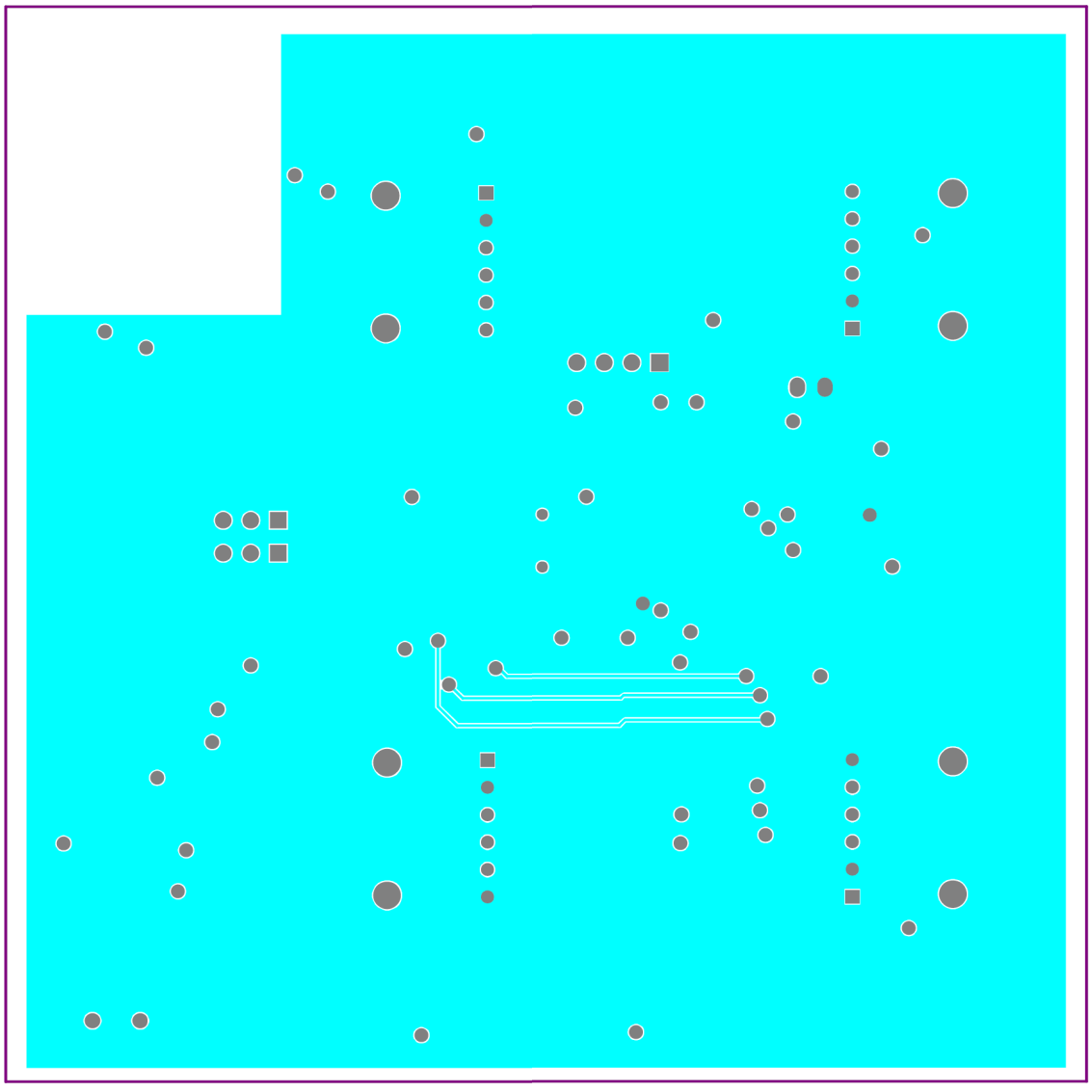
Obrázek C.1: Kompletní schéma detektoru sirény



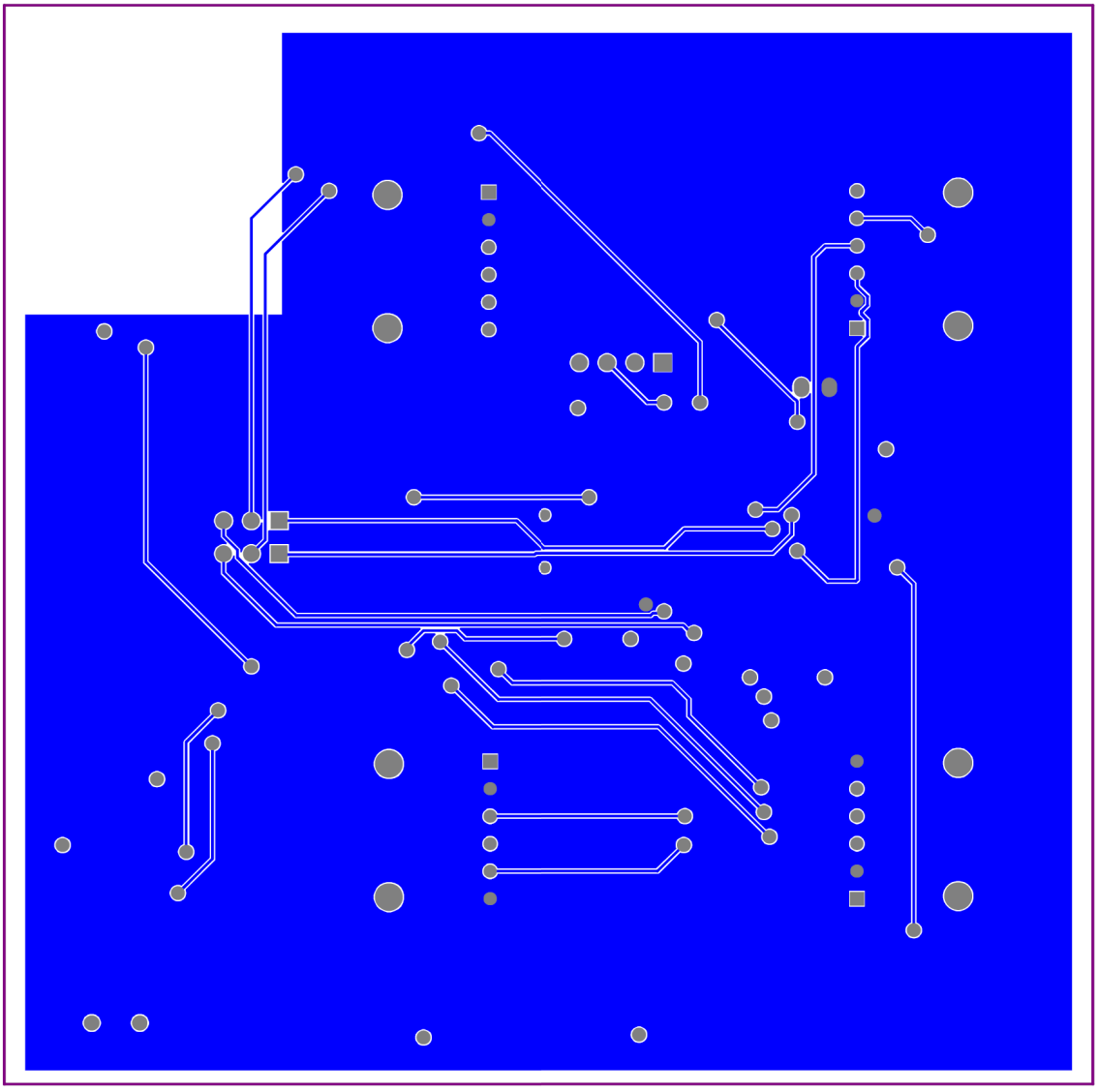
Obrázek C.2: Horní vrstva s popiskami



Obrázek C.3: První vnitřní vrstva

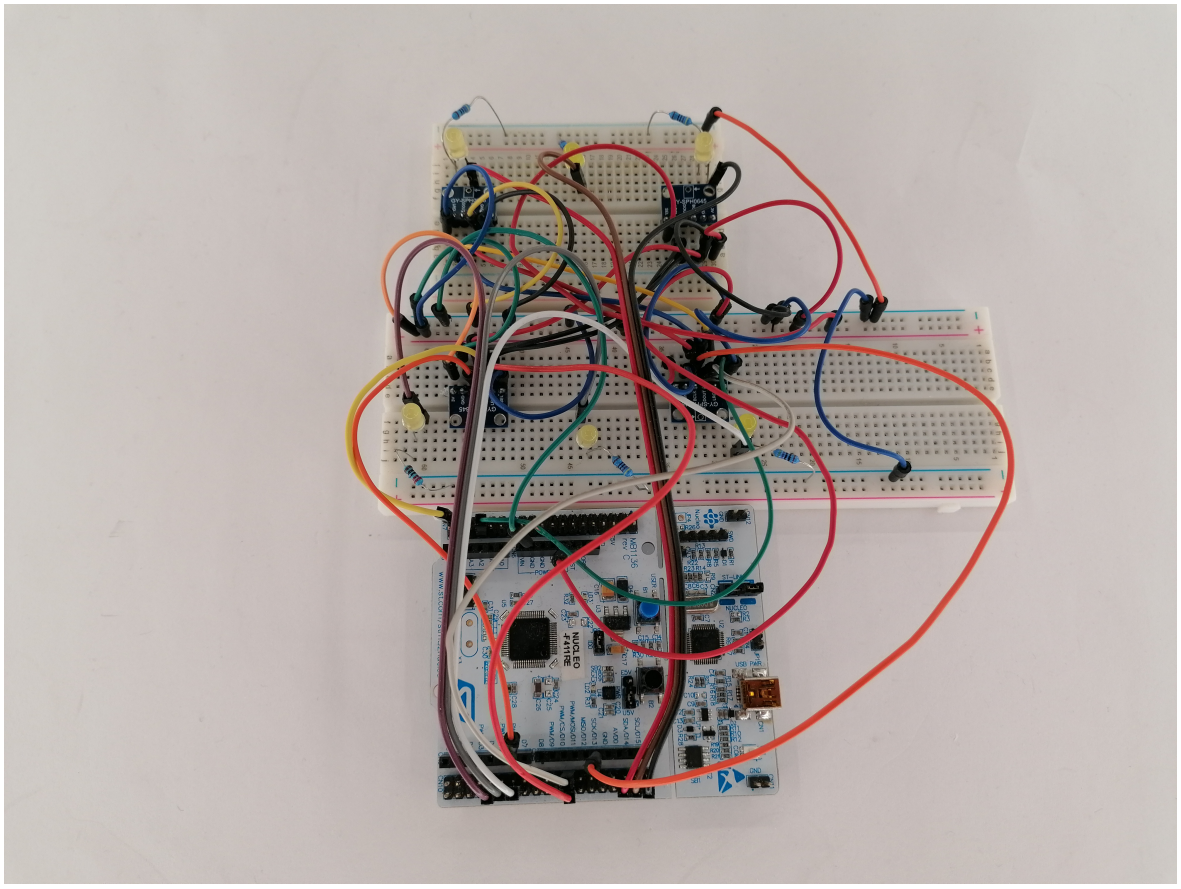


Obrázek C.4: Druhá vnitřní vrstva



Obrázek C.5: Spodní vrstva

Příloha D
Zapojení na nepájivém poli



Obrázek D.1: Nucleo-STM32F411RE s mikrofony v nepájivém poli

Příloha E
Obsah přiloženého CD

/
- `DP_Funderak_Michael_2023.pdf` ..... Diplomová práce v PDF formátu
- `Altium_project` ..... Návrh DPS v programu Altium Designer
- `DP_Microcontroller_project` .... STM32CubeIDE projekt pro mikrokontrolér
- `Python_scripts_MFCC_generation` .. Skripty v Pythonu pro generování MFCC
- `Trained_models` ..... Natrénované modely neuronových sítí
- `Training_project` ..... Projekt na trénování neuronových sítí