Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

DIPLOMOVÁ PRÁCE

Rozpoznávání hlasových příkazů na mikroprocesoru

PLZEŇ, 2018 Bc. DAVID BENEŠ

Čestné prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne:_____ .....

Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce Ing. Luboši Šmidlovi, PhD. za poskytnutí cenných rad a vstřicný přístup během zpracování této práce.

Abstrakt

Tato práce řeší problematiku rozpoznávání hlasových příkazů na mikroprocesorech. Mikroprocesor je víceúčelová integrovaná součástka obsahující výpočetní jádro, paměť a veliké množství periférií. Problematika rozpoznávání hlasových příkazů je řešena relativně jednoduchými metodami pro vytvoření příznakových vektorů v reálném čase. Po detekci hlasu se spustí porovnávání hlasového příkazu s nahranými vzory, pomocí metody dynamic time warping (DTW). Výsledek je poté možné zpracovat, například rozsvícením kontrolky nebo posláním dat pomocí sériové linky.

Abstract

This work is trying to solve a problem of voice command recognition on microprocessors. The microprocessor is a multipurpose integrated component that includes a computational core, memory, and many of peripherals. Problem of voice command recognition is solved by relatively simple methods for creating real-time feature vectors. After detection of voice, a comparison of a voice command with uploaded patterns with Dynamic Time Deformation (DTW) starts. The result can then be processed, for example by turning on the light or send information over usart.

Obsah

Úvod

Rozpoznávání řeči je stále rozvíjející se odvětví umělé inteligence. Z historického hlediska, kdy se první experimenty datují již v roce 1952 v Bellových laboratořích, však došlo k velikému pokroku. Od jednoduchého rozpoznávání číslic závislé na řečníkovi, se přesunula oblast rozpoznávání na úroveň komplexní mluvy s podporou gramatiky, které je nezávislé na řečníkovi. Jeden z největších přínosů, byla aplikace skrytých markovových modelů (HMM), které využívají pravděpodobnostní přístup pro rozpoznávání. Dnes se stále více nasazují neuronové sítě (DNN), které dosahují vyšší přesnosti rozpoznávání. Více zajímavých informací lze nalézt v literatuře [1.1] a [1.2]. Rozpoznávání řeči (ASR) je možné nalézt na několika platformách jako jsou telefony, počítače, automobily atd. Problém s kvalitnějším a složitějším systémem pro ASR je zvyšující se náročnost na pamět a výpočetní výkon zařízení na kterém systém běží. V případě systému běžícího na telefonech je jisté, že ASR ukládá data na vzdálených serverech. To může být pro některé aplikace nepřípustné, stejně tak potřeba stálého připojení k internetu při využití vzdáleného výpočetního stroje k výpočtům.

Tato práce se zabývá vytvořením jednoduchého systému rozpoznávání řeči od hardwaru až po software pro aplikaci na mikrokontrolérech. Mikrokontrolér je v dnešní době nejrozšířenější typ procesoru na světě. Díky integrované paměti, výpočetního jádra, periferiím a velmi nízké spotřebě elektrické energie je možné tuto součástku nalézt téměř všude, v různých provedeních a výkonových třídách.

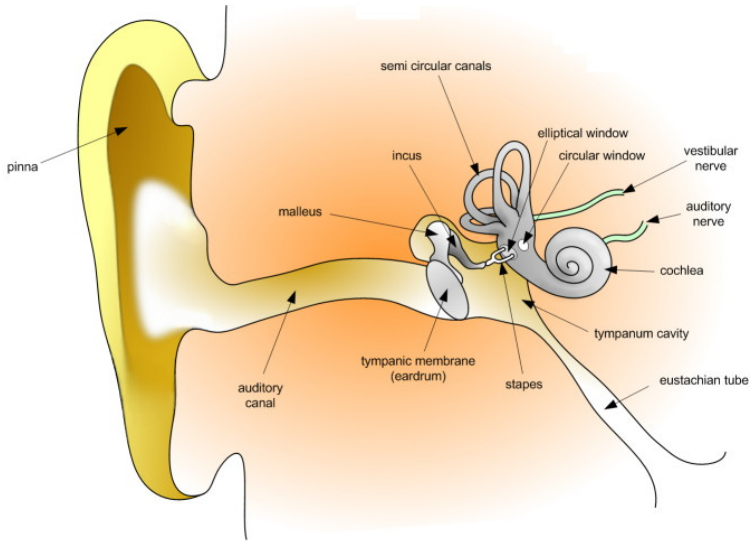
Cílem této práce je vytvoření ASR běžící na mikroprocesoru. Aplikace starších metod ASR optimalizované tak, aby běžely co nejefektivněji s ohledem na omezené zdroje hardwaru. S přechodem na výkonější typ mikroprocesoru by se mohlo otevřít více možností se stejným principem výpočtů. Výsledný systém by měl být nezávislý na externích datových a výpočetních zdrojích.

2. Teorie

Mluvená řeč je jedna z nejstarších forem komunikace mezi lidmi. Slouží k vyjádření myšlenek, požadavků a sdělení. Z tohoto důvodu je snaha vytvořit dialogový systém schopný zpracovávat řeč počítačem tak, abychom s ním mohli komunikovat podobně, jako s jinou osobou. Tato problematika je ovšem velmi složitá, protože se řeč za tisíciletí velmi rozvinula a stále se vyvíjí. Každý člověk má jinou formu vyjadřování, jinou barvu hlasu, jinou slovní zásobu či jiné nářečí. Člověk rozpozná rozdíly v mluvě většinou bez větších problémů a to nejen díky znalostem, ale i zkušenostem a odhadu. V nejhorším případě jsme alespoň schopni kvalitativně odhadnout o co se může v řeči jednat. Sluch člověka je stejně jako citlivost mikrofonu zatížena okolním šumem. Nicméně člověk, nejenže rozpozná hlas v zašuměném prostředí, ale je schopen i rozpoznat o jaký šum se jedná. Například pozná zvuk motoru, hudbu v pozadí, hučení ventilátoru, smích nebo jiné zvuky netýkající se informace obsažené v řeči.

2.1 Zvuk a jeho záznam

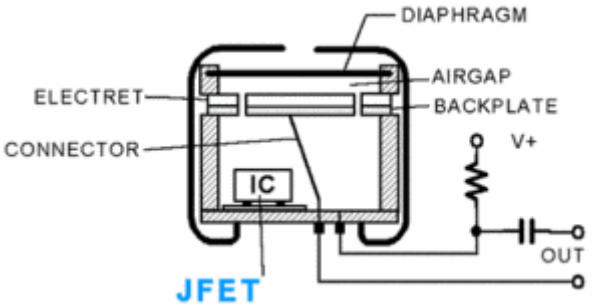
Zvuk není nic jiného než mechanické vlnění šířicí v prostředí. Částice v prostředí (vzduch, kapalina, pevná látka) začínají kmitat při interakci s akustickou vlnou a způsobují další šíření akustické informace do protředí. Samotné prostředí se tedy přímo podílí na zkreslení informace nesené akustickým kanálem. Forma zkreslení může být: útlum amplitudy, vznik ozvěn či tvorba sekundárních akustických signálu např. rozechvěním těles atd.



Obrázek 2.1: Vnitřní struktura sluchového ústrojí. [2.1]

Frekvenční rozsah, který dokáže běžný člověk vnímat je v rozsahu od 16Hz do 20kHz [2.2]. Nižší frekvence je označována jako infrazvuk a vyšší frekvence je ozvačována jako ultrazvuk. Nejdůležitější složky lidského hlasu se nacházejí v oblasti od 1kHz do 3kHz. V tomto frekvenčním pásmu je sluch člověka nejcitlivější. Vyšší frekvence se podílejí na barvě hlasu, což je vhodné pro rozpoznávání řečníka.

Člověk vnímá zvuk následujícím způsobem. Zvuk projde zvukovodem a akustické vlny narážejí do bubínku, ten se úměrně energii signálu rozechvěje a vzniklé vibrace se přenášejí přes kladívko (malleus), kovadlinku (incus) a třmínek (stapes) do hlemýždě (cochlea), ze kterého je poté nervovou cestou vedena informace přímo do mozku. Kladívko, kovadlinka a třmínek jsou jedny z nejmenších kůstek v lidském těle. Hlemýžď by se dal přirovnat ke spektrografickému analyzátoru, neboť výstupem je intenzita jednotlivých frekvenčních složek obsažených ve zvuku. Vnitřní struktura sluchového ústrojí je zobrazena na obrázku 2.1.



Obrázek 2.2: Schéma vnitřku elektretového mikrofonu. [2.3]

Pro snímání zvuku elektronickým zařízením slouží mikrofon. První mikrofon byl představen již v roce 1877 vynálezcem Emiliem Berlinerem. Mikrofon převádí akustický signál na elektrický. Způsob převodu závisí na jeho konstrukci. Existuje několik typů, některé se již nepoužívají: kondenzátorový mikrofon, elektretový mikrofon, dynamický mikrofon, páskový mikrofon, uhlíkový mikrofon a piezoelektrický mikrofon. Nejpoužívanějším typem je elektretový mikrofon z důvodu jednoduché konstrukce, ceny a velmi malých rozměrů (v porovnání s ostatními typy). Je také využit jako zvukový senzor v této práci.

Elektretový mikrofon se skládá z vodivé základny, elektretové membrány a interního předzesilovače nejčastěji v podobě FET tranzistoru[1]. Akustický signál, který naráží na membránu způsobí její rozkmit. Tento rozkmit má za následek napěťové změny na základně. Tyto napěťové změny poté mění elektrickou vodivost FET tranzistoru. Mikrofon musí být připojen na zdroj napětí a svojí funkcí mění odpor dle aktuální frekvence a amplitudě působící na jeho membránu. Na

1 FET tranzistor – field-effect transistor. Známé též jako unipolární transistor. Transistor využívající elektrické pole k ovlivňování vlastnosti přechodu. [2.4]

obrázku 2.2 je znázorněna vnitřní struktura elektretového mikrofonu v provedení backplate. Diaphgram je membrána, electret je elektretová vložka a backplate je vodivá základna. Mikrofon je tedy připojen na napětí s předřadným neměnným odporem. Výstupem tohoto zapojení je spojitý signál v podobě napětí úměrné úbytku napětí na mikrofonu.

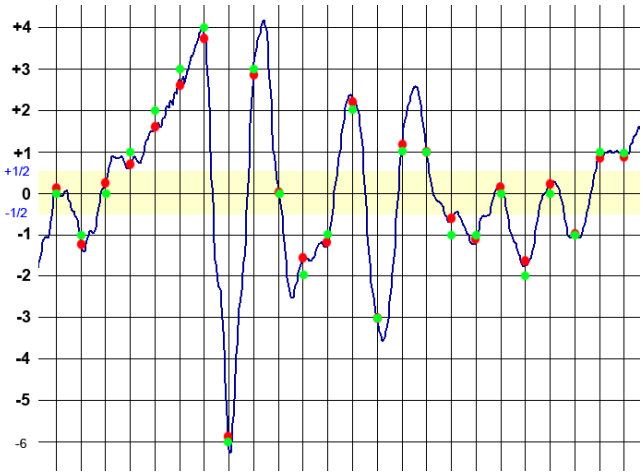
2.2 Digitální záznam zvuku

Záznam akustického signálu v digitální podobě má velmi zásadní význam pro budoucí zpracování. Není technologicky možné zaznamenat audio signál spojitě v číslicové podobě. Z tohoto důvodu se analogový signál vzorkuje s danou periodou T, tento proces se nazývá vzorkováním. Podle Shannonova teorému by při vzorkování signálu měla být vzorkovací frekvence F_S minimálně dvakrát větší, než je největší frekvenční složka obsažena v signále o frekvenci F_D tedy měl by být splněn vztah (2.1).

$$F_S \geq 2 * F_D \quad (2.1)$$

Po každé periodě $T=1/F_S$ dojde ke záznamu vzorku (nejčastěji AD převodníkem) a analogová hodnota se převede na číslicovou. Tomuto procesu se říká kvantizace. Při kvantizaci dochází k chybě, obecně nazývaná kvantizační chyba. Tato chyba je způsobena omezeným rozlišením AD převodníků. Čím větší rozlišení AD převodníku, tím menší kvantizační chyba. Kvantizační šum je vyjádřený v decibelech, udávající poměr užitečného signálu k šumu, je to chyba jednotlivých vzorků od reálného signálu. Vztah je vyjádřen vzorcem (2.2) a hodnota N vyjadřuje rozlišení AD převodníku.

$$SNR_{AD} = 20 \log 2^N \approx 6{,}02\, N\, [dB] \quad (2.2)$$



Obrázek 2.3: Ukázka vzorkování a kvantizace [2.4]

Na obrázku 2.3 je znázorněn průběh signálu, který se vzorkuje ideálním AD převodníkem. Modrá křivka je analogová (reálná) podoba signálu. Červené body v grafu jsou hodnoty, které by

v ideálním případě měly být naměřeny. Zelené body jsou naměřené celočíselné hodnoty. Horizontální přímky jsou jednotlivé kvantizační úrovně. Čím větší rozlišení AD převodníku, tím vyšší hustota horizontálních přímek. Vertikální přímky jsou okamžiky vzorkování. Zvýšíme-li vzorkovací frekvenci, zvýšíme hustotu vertikálních přímek.

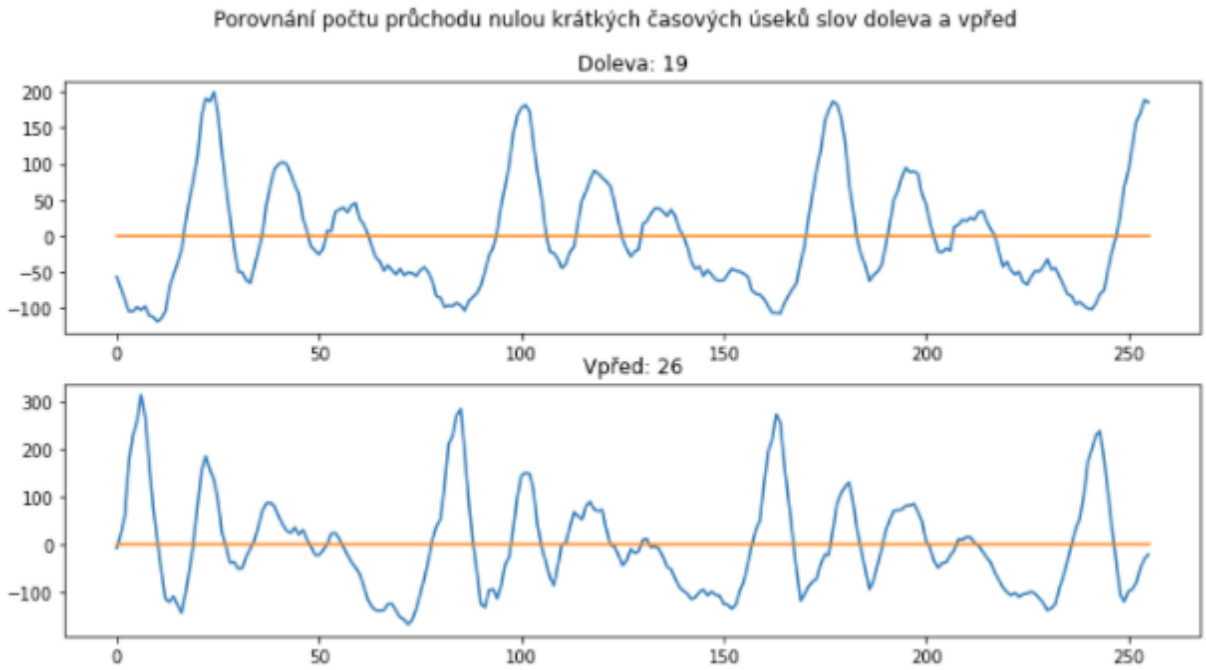
I přesto, že vstupní signál vždy prochází dolnopropustným filtrem, tak se v reálném případě v každé aplikaci bude vždy vyskytovat vysokofrekvenční šum. Tento šum má velmi malou amplitudu, takže by neměl ovlivnit kvatizační úroveň vzorku. Ovšem v případě, kdy by byl AD převodník s opravdu velmi vysokým rozlišením, mohl by tento vysokofrekvenční šum zatížit výstupní signál, takže by bylo nutné zvýšit vzorkovací frekvenci.

2.3 Časová analýza

Akustický signál v časové oblasti nenese tolik informací o řeči jaké bychom potřebovali. Na základě dat signálu v časové oblasti můžeme zjistit délku promluvy, energii signálu, korelaci signálu a střední počet průchodu nulou. Délka promluvy může být užitečná při porovnávání se vzorem. Vezmeme-li například dvě slova: ano a velkorysost. Časová délka slova ano bude znatelně kratší, než délka slova velkorysost. Počet průchodů nulou je počet, kolikrát signál projde 0. Tato hodnota může být využita k porovnání dvou slov o stejné délce. Na obrázku 2.4 je znázornění porovnání krátkých časových úseků dvou slov: doleva a vpřed (modrá křivka) a nuly (žlutá křivka). Při aplikaci algoritmu detekce počtu průchodu signálu nulou zjistíme, že krátké časové úseky těchto dvou slov mají rozdílné počty průchodu nulou.

Energie signálu bývá velmi často využita jako hodnota pro aktivaci automatického rozpoznávání řeči. Nebo-li detekce okamžiku, kdy má význam spustit rozpoznávání. Autokorelace signálu může být využita k vyhodnocení základní frekvence hlasivek nebo podobnosti signálu. Autokorelace je významným nástrojem k identifikaci periodičnosti signálu. Pokud je signál periodický s periodou T, tak autokorelační funkce dosahuje svých maxim v periodách n=T, 2T, 3T, ...

Obecně kromě energie signálu a autokorelace je analýza v časové oblasti velmi omezená. Při analýze se signál rozděluje na mikrosegmenty. Mikrosegment je získán dělením signálu na několik krátkých úseků pomocí definovaného okénka. Často užívaná okénka jsou pravoúhlé okénko, Hammingovo okénko, Blackmanovo okénko atd. V případě pravoúhlého okénka se signál přenásobí hodnotou 1 je-li vzorek v daném rozsahu okénka a 0 je-li mimo. Ignorováním hodnot mimo okénko nám tedy zůstanou data pouze uvnitř okénka → výsledný mikrosegment.



Obrázek 2.4: Porovnání 32ms úseků slov doleva a vpřed

Matematický zápis výše zmíněného mikrosegmentu lze zapsat pomocí následujícího vzorce (2.3), kde Q_n je krátkodobá charakteristika, s(k) je vzorek signálu, w(n) je okénková funkce. Okénkové funkce mají tvar v podobě vztahů (2.4) pro pravoúhlé okénko a (2.5) pro Hammingovo okénko. Délka mikrosegmentu by se měla pohybovat ideálně mezi hodnotami 20ms až 35ms [2.5]. Tato délka mikrosegmentu je taková z důvodu, protože při kratším úseku můžeme přijít o informace o pomalých změnách signálu. Naopak při delším úseku můžeme zahrnout do analyzovaného mikrosegmentu i informace příliš rychlých změn.

$$Q_n = \sum_{k=-inf}^{inf} s(k)\,w(n-k) \quad (2.3)$$

$$w(n) = \begin{matrix} 1 & pro\ 0 \le n \le L-1 \\ 0 & pro\ ostatní \end{matrix} \quad (2.4)$$

$$w(n) = \begin{matrix} 0{,}54 - 0{,}46\cos(2\pi n/(L-1)) & pro\ 0 \le n \le L-1 \\ 0 & pro\ ostatní \end{matrix} \quad (2.5)$$

V praxi se signál rozdělí na mikrosegmenty a poté se analyzuje. Výše zmíněné přístupy by měly následující tvar. Vztah (2.6) je výpočet energie daného mikrosegmentu. Vztah (2.7) je výpočet průchodu nulou v daném mikrosegmentu, sgn() má význam funkce signum (pro hodnoty x<0 je hnodnota -1 a pro hodnoty x>=0 je hodnota 1). Vztah (2.8) je výpočet autokorelační funkce. Ve všech těchto vztazích index n vyjadřuje index mikrosegmentu a index k v sumě vyjadřuje index vzorku v daném mikrosegmentu.

$$E_n = \sum_{k=-inf}^{inf} Q_n(k)^2 \quad (2.6)$$

$$Z_n = \sum_{k=-inf}^{inf} \left| sgn(Q_n(k)) - sgn(Q_n(k-1)) \right| \quad (2.7)$$

$$R_n(m) = \sum_{k=-inf}^{inf} Q_n(k) * Q_n(k+m) \quad (2.8)$$

Energii mikrosegmentu můžeme využít pro start rozpoznávání. Problém nastává se zvoleným prahem. Může být konstantní, což zhoršuje vlastnosti ve hlučném prostředí, nebo může být dynamický na základě průměrné energie doposud vypočtených mikrosegmentů či kombinace obou dvou. Dynamický práh je výhodnější pro aplikaci v neznámém akustickém prostředí, může však nastat problém v okamžiku pomalu zvyšující se energie.

2.4 Frekvenční analýza

Jak bylo naznačeno v kapitole 2.1, tak za biologický proces vnímání zvuku odpovídá orgán cochlea, který zvuk zpracuje ve frekvenční oblasti. Důvod zpracovávání zvuku ve frekvenční oblasti je ten, že spektrum obsahuje mnohem více užitečných informací. Spektrum se v diskrétním přístupu vypočte pomocí Fourierovy transformace, která nám spojitý signál převede na jednotlivé složky frekvenčního spektra.

Obecně se využívá diskrétní Fourierova transformace z důvodu vzorkování pomocí číslicových systémů, v této práci tomu není jinak. Většinou se využívá rychlá Fourierova transformace (fast fourier transform – FFT), což je efektivní typ výpočtu diskrétní Fourierovy transformace. Vstupní i výstupní hodnoty se nacházejí v komplexní oblasti hodnot. Běžná formulace je dle vzorce (2.9), jedná se o DFT (discrete Fourier transform).

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi \frac{n}{N}}; k = 0,.., N-1 \quad (2.9)$$

V předchozím vztahu jsou hodnoty $\underline{X_k}$ (Fourierův obraz) a $\underline{x_n}$ (data signálu) komplexní čísla. Ačkoliv většinou $\underline{x_n}$ jsou reálná čísla získaná pomocí AD převodníku. Výpočetní složitost DFT je $O(N^2)$[2]. DFT zahrnuje N^2 komplexního násobení a N(N-1) komplexního sčítání. Za předpokladu, že se omezíme na data o velikosti 2^N a upravíme vztah (2.9), můžeme hovořit o FFT tkz. 2-radix Cooley-Turkey algoritmus. Tato FFT zpracuje $(N/2)\log_2(N)$ komplexního násobení za

2 O(N) je vyjádření asymptotické výpočetní složitosti. Udává estimovanou dobu zpracování algorithmu na základě velikosti dat N. [4]

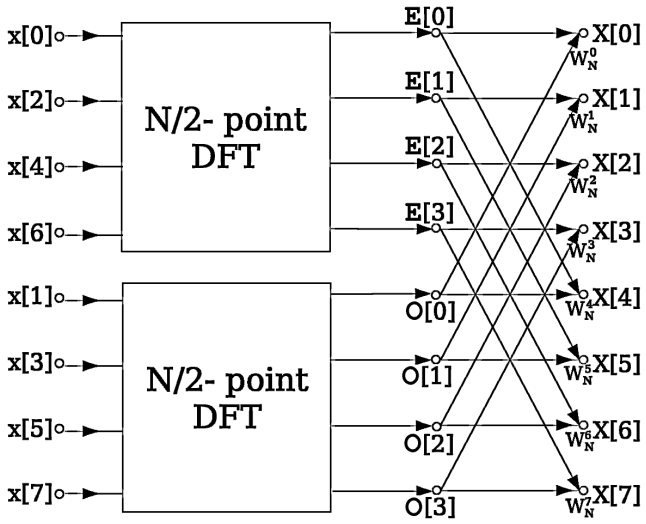
předpokladu ignorování násobení hodnotou 1 a $N\log_2(N)$ komplexního sčítání. Výsledná výpočetní složitost je o řád nižší, než v případě DFT a to O(N log (N)). 2-radix algoritmus rozdělí výpočet na liché a sudé prvky. Tedy vztah (2.9) lze přepsat na (2.10).

$$X_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2\mathrm{m}} e^{-\frac{2\pi i}{N}(2\mathrm{m})k} + \sum_{m=0}^{\frac{N}{2}-1} x_{2\mathrm{m}+1} e^{-\frac{2\pi i}{N}(2\mathrm{m}+1)k} \quad (2.10)$$

Úpravou vztahu (2.10) přeznačením na E_k (sudé prvky – levá suma) a O_k (liché prvky pravá suma) a vytknutím $e^{-\frac{2\pi i}{M}k}$ před pravou sumu získáme upravený vztah (2.11). Pro urychlení výpočtu je možné a doporučené předvypočítat hodnoty exponentů do lookup tabulek. Vztah lze ještě přepsat na upravený vztah, kde je výsledkem využití již spočtených hodnot (2.12). Na obrázku 2.5 je nakreslen tkz. butterfly, jedná se o diagram znázorňující průběh (nebo cestu) výpočtu FFT na datech o velikosti 8. Je viditelné, že na vstupu jsou hodnoty s přeindexovaným vstupem. Tato data se rozdělí na polovinu a vypočtou se dvě DFT a poté se postupuje po uzlech v diagramu. Výstupem je správně seřazená fourierova transformace v komplexních číslech.

$$X_k = E_k + e^{-\frac{2\pi i}{N}k} O_k \; ; E_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2\mathrm{m}} e^{-\frac{2\pi i}{N}(2\mathrm{m})k} \; ; O_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2\mathrm{m}+1} e^{-\frac{2\pi i}{N}(2\mathrm{m})k} \quad (2.11)$$

$$\begin{aligned} X_k &= E_k + e^{-\frac{2\pi i}{N}k} O_k \\ X_{k+\frac{N}{2}} &= E_k - e^{-\frac{2\pi i}{N}k} O_k \end{aligned} \quad (2.12)$$



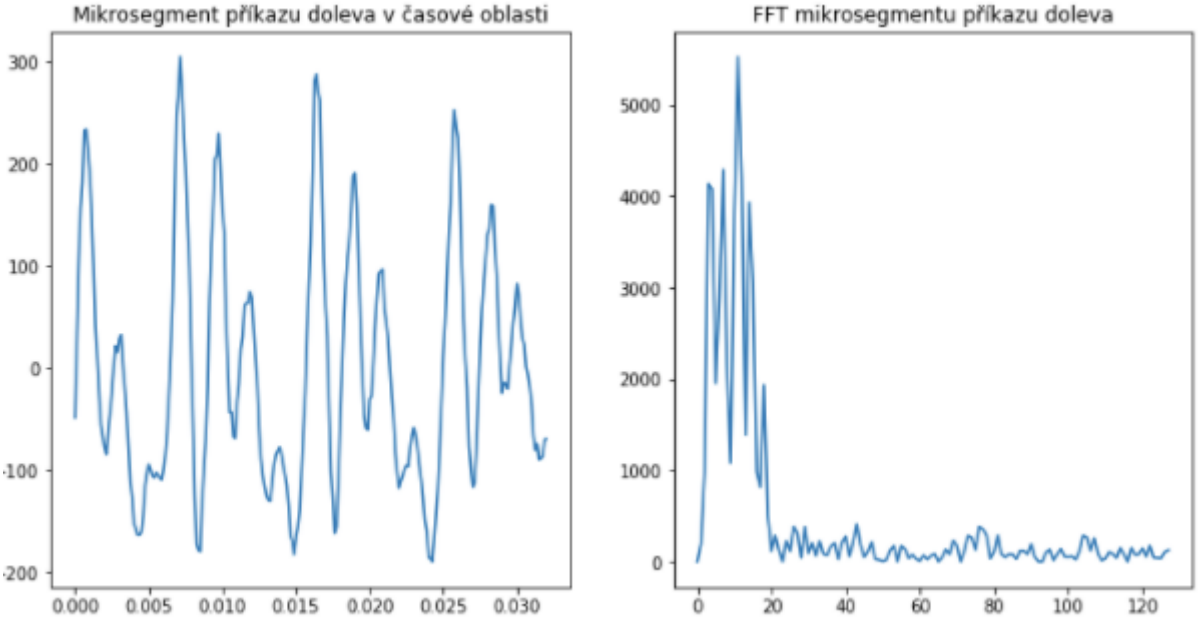
Obrázek 2.5: Diagram výpočtu pro N=8 radix-2 FFT [2.6]

Pro ilustraci je na obrázku 2.6 provedena fourierova transformace jednoho mikrosegmentu příkazu "vpřed". Jedná se o reálná data získaná přímo z mikroprocesoru. Okénková funkce je typu pravoúhlé okénko. Znázorněné hodnoty FFT jsou odmocninou energie jednotlivých frekvenčních úrovní a vypočte se pomocí vztahu (2.13). V grafu FFT je zobrazena pouze polovina výsledku.

$$P(k)=\sqrt{X_r(k)^2+X_i(k)^2} \quad (2.13)$$

Hodnota X_r představuje reálnou část komplexního čísla a X_i imaginární část komplexního čísla.

Na obrázku 2.6 je viditelné, že dominantní frekvenční složka v daném mikrosegmentu je okolo frekvence f_{dom} = 17*4096/128 = 531 Hz a ve frekvenčním spektru se pak vyskytuje několik minoritních špiček (400Hz,1376Hz,...).



Obrázek 2.6: Analýza signálu krátkého úseku promluvy

2.5 Vnímání zvuku

Ve většině přístupů umělé inteligence je snaha napodobit pracovní činnost, popřípadě logické úvahy člověka v dané oblasti, kde se má UI[3] uplatnit. Pro rozpoznávání hlasu není tento přístup jiný. Člověk vnímá zvuk odlišným způsobem než-li počítač. V některých případech to může být výhodnější. Pomocí frekvenční analýzy zvuku zaznamenaného mikrofonem, můžeme stanovit například rezonanční frekvence různých soustrojí, popřípadě analyzovat zvuk mnohem detailněji než by zvládl jakýkoliv člověk.

Záznam zvuku strojem má tu výhodu, že v záznamu neexistuje žádný subjektivní dojem, citlivost na různých frekvencích bude vždy stejná (za předpokladu neměnící se teploty a degradace

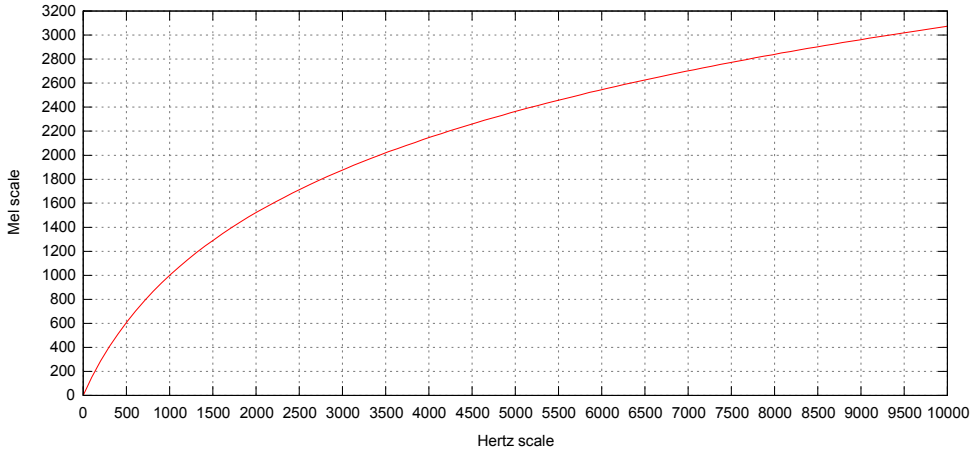
3 UI – umělá inteligence

zařízení stářím), časový odstup mezi vzorky bude rovnoměrný atd. Nicméně se zvyšující se kvalitou zaznamenaného zvuku (hlavně v oblasti vzorkovací frekvence) velmi roste výpočetní náročnost pro rozpoznávání příkazů/slov/slabik. Jak již bylo řečeno, nejvíce významných příznaků řeči lze získat z frekvenční analýzy. Nicméně při zachování 20ms až 35ms mikrosegmentu a využití FFT se počet analyzovaných frekvencí v mikrosegmentu se zvyšujícím vzorkováním zvyšuje.

Frekvence	4000Hz	8000Hz	16000Hz	32000Hz	64000Hz
Délka mikrosegmentu	128 vzorků	256 vzorků	512 vzorků	1024 vzorků	2048 vzorků
Počet frekvencí	64 frekvencí	128 frekvencí	256 frekvencí	512 frekvencí	1024 frekvencí

Tabulka 2.1: Hodnoty FFT pro různé vzorkovací frekvence

V předchozi tabulce jsou vypsané různé vzorkovací frekvence (nejen běžně používané) včetně délky mikrosegmentu v počtu vzorků ze signálu. Počet frekvencí vyjadřuje polovinu délky vektoru, který získáme pomocí FFT algoritmu. Pokud bychom tedy vzorkovali signál vysokou frekvencí (například 64kHz), získali bychom velmi mnoho údajů i o frekvencích, které jsou pro člověka neslyšitelné.

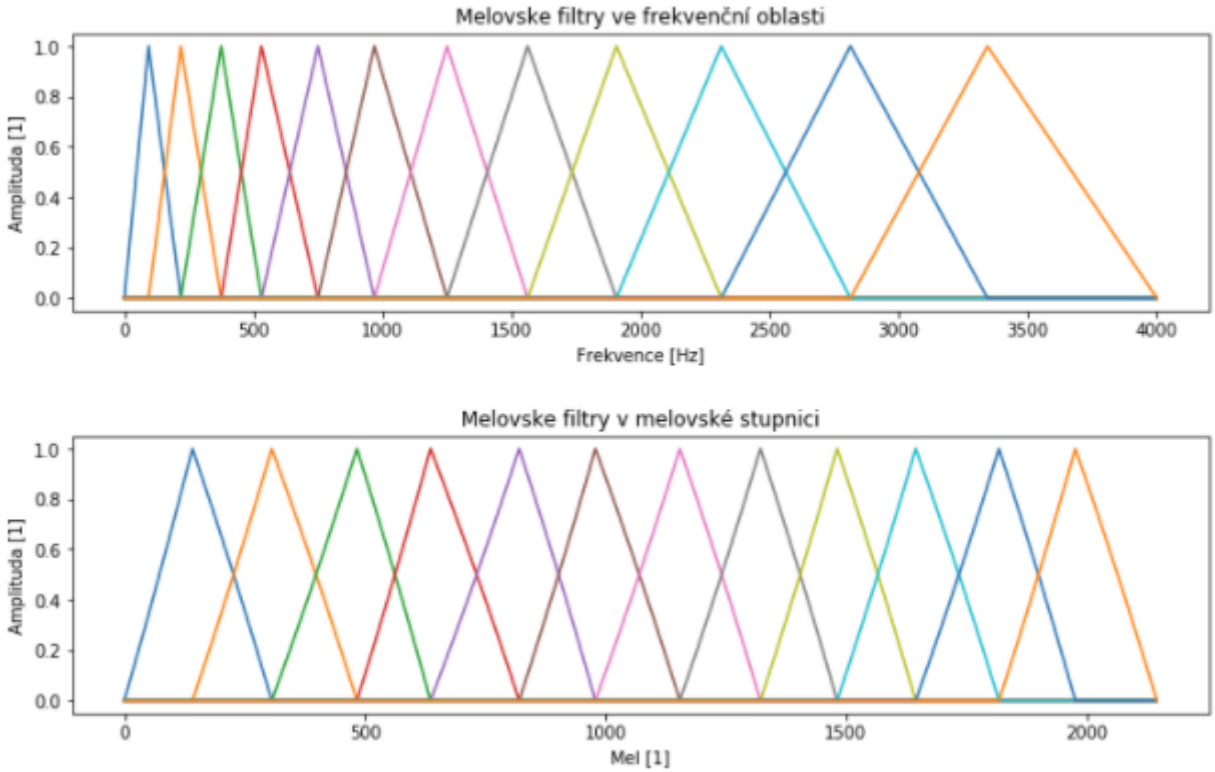


Obrázek 2.7: Vztah mezi frekvencí a melovskou stupnicí [2.7]

Nicméně tolik parametrů je z hlediska informace pro rozpoznávání řeči zbytečných. V reálných aplikacích je snaha rozumně redukovat příznakové vektory vhodnými metodami. Tyto metody pro rozpoznávání řeči vycházejí z principu vnímání zvuku člověkem. Člověk nemá absolutní vjem o rozdílu mezi dvěma frekvencemi. Tento fakt je způsoben fyziologií člověka. Pokud tedy budeme vnímat frekvenci o hodnotě 2000Hz, tak budeme mít dojem, že frekvence o dvojnásobné hodnotě bude cca 8000Hz. Na základě experimentů vznikla Melova stupnice, která převádí vnímanou frekvenci a reálnou frekvenci. Mezi frekvencí a Melovou stupnicí je závislost popsaná vztahem (2.14).

$$F_{mel} = 2595 \log\left(1 + \frac{F_{Hz}}{700}\right) \quad (2.14)$$

Na základě Melovy stupnice můžeme relativně jednoduše redukovat příznakové vektory z velikosti odpovídající frekvenčním pásmům získané pomocí FFT na požadovaný počet příznaků pomocí trojúhelníkových filtrů rovnoměrně rozložené na Melově stupnici. Převedením těchto filtrů Melovy stupnice do frekvenční oblasti dojde k „deformaci“. Na základě počtu filtrů se vypočtou jejich středové frekvence a velikost [2.9].



Obrázek 2.8: Znázornění filtrů ve frekvenční oblasti a v Melově stupnici

Na předchozím obrázku je znázorněné rozdělení 12 Melových filtrů. Filtry jsou rozděleny rovnoměrně v Melově stupnici a při převodu zpět do frekvenční oblasti dojde k deformaci na horizontální ose. K této transformaci se využívá vztah (2.15).

$$F_{Hz} = 700 * \left(e^{\left(\frac{F_{mel}}{1125}\right)} - 1\right) \quad (2.15)$$

Při výpočtu filtrů tedy postupujeme tak, že nejdříve zvolíme počet filtrů. Tomuto počtu bude odpovidat i velikost příznakového vektoru. Musíme znát vzorkovací frekvenci. Polovinu této frekvence převedeme do Melovy stupnice a bude vyjadřovat maximální hodnotu. Poté pomocí jednoduché aritmetiky rozdělíme výsledný úsek na M dílků tak, aby vznikly rovnoramenné trojúhelníky. Jednotlivé vrcholy lze vypočítat pomocí vztahu (2.16) a (2.17). Vztah (2.18) je pro výpočet délky filtru v Melově stupnici a vztah (2.17) je pro stanovení pozic vrcholů filtrů v Melově

stupnici. Po stanovení filtrů v Melově stupnici se pomocí vztahu (2.15) převedou tvary filtrů do frekvenční oblasti.

$$Mel_L = \frac{Mel_{max}}{M+2},\ Mel_{max} - vzorkovací\ \ frekvence\ v\ mel\ stupnici\ ,\ M - počet\ \ filtrů \quad (2.16)$$

$$H_{mel}(k) = k * Mel_L \quad (2.17)$$

	Minimum	Maximum	Velikost filtru
Frekvence [Hz]	0 Hz	4000 Hz	různé
Mel stupnice [1]	0 mel	2142,27 mel	153 mel

Tabulka 2.2: Zabulka rozsahů hodnot filtrů

Následující vztah je modelem pro výpočet jednotlivých filtrů podle pozice. Nejčastěji jako index odpovídající frekvenci získané pomocí FFT. $H_m(k)$ je vypočtená hodnota filtru na dané pozici k a m je index filtru. Hodnota f(..) je index vrcholu filtru.

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \le k \le f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \le k \le f(m+1) \\ 0 & k > f(m+1) \end{cases}$$

Vztah 2.1: Model výpočtu filtrů [7]

Následující tabulka je zkráceným znázorněním hodnot prvních třech Melových filtrů na jednotlivých indexech výstupu FFT pro délku signálu 256, 12 Mel filtrů o vzorkovací frekvenci 8kHz. Nejjednodušší výpočet vystupůu filtrů je tedy konvoluce zvoleného filtru s vektorem získaným pomocí FFT.

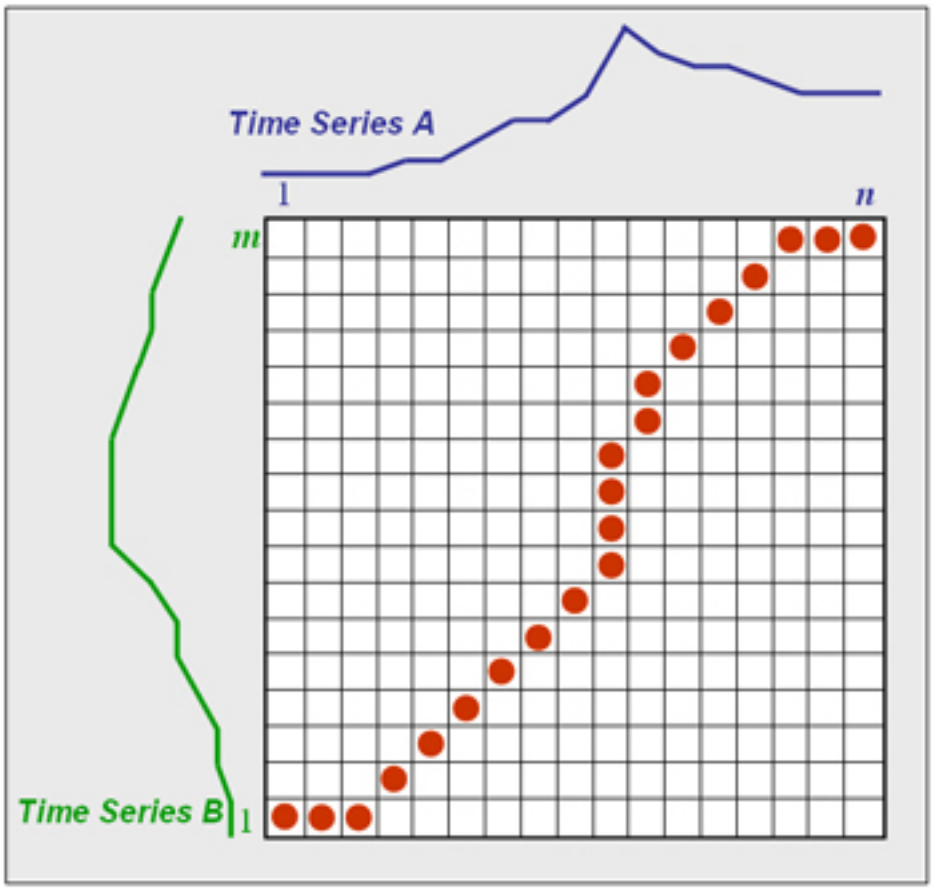
Filter	Index hodnoty po provedení FFT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15->
0	0	0,33	0,67	1	0,75	0,5	0,25	0	0	0	0	0	0	0	0	...
1	0	0	0	0	0,25	0,5	0,75	1	0,8	0,6	0,4	0,2	0	0	0	...
2	0	0	0	0	0	0	0	0	0,2	0,4	0,6	0,8	1	0,8	0,6	...

Tabulka 2.3: Hodnoty filtrů na daných indexech

2.6 Porovnávání příznakových vektorů

Pro porovnání dvou signálů pomocí příznakových vektorů lze využít velmi kvalitně dynamic time warping algoritmus (DTW). Tento algoritmus nám zajistí nalezení podobnosti dle zvolené míry a současně oba dva signály nemusejí mít stejnou délku. Většinou se výsledná podobnost nebo-li cena znázorňuje ve 2D poli a ideální cesta by se měla nacházet na diagonále. DTW je velmi efektivní metoda pro porovnávání dvou datových sekvencí. Podoba algoritmu je v kapitole 4.4.

Následující obrázek zobrazuje nalezení nejkratší cesty mezi dvěma signály (nebo sekvencemi dat v podobě příznakových vektorů).



Obrázek 2.9: Cesta DTW [2.10]

Literatura

[2.1] *www.wikiskripta.eu* [online]. [cit. 27.1.2018]. Dostupné na WWW: https://upload.wikimedia.org/wikipedia/commons/7/7c/HumanEar.jpg

[2.2] Člověk a zvíře: Sluchový rozsah | Eduportál Techmania. *Eduportál* | *Eduportál Techmania* [online]. Copyright © Techmania Science Center, o.p.s. [cit. 05.04.2018]. Dostupné na WWW: http://edu.techmania.cz/cs/katalog/clovek-zvire/316/sluchovy-rozsah

[2.3] *wiki.metropolia.fi* [online]. [cit. 27.1.2018]. Dostupné na WWW: http://hades.mech.northwestern.edu/images/thumb/8/80/ElectretMicrophone.gif/300px-ElectretMicrophone.gif

[2.4] ŠERÝCH, Jakub. *cs.wikipedia.org* [online]. [cit. 27.2.2018]. Dostupné na WWW: https://upload.wikimedia.org/wikipedia/commons/e/ed/Kvantov%C3%A1n%C3%AD.png

[2.4] Field-effect transistor - Wikipedia. [online]. [cit. 27.1.2018] Dostupné na WWW: https://en.wikipedia.org/wiki/Field-effect_transistor

[2.5] PSUTKA, Josef. MÜLLER, Luděk. MATOUŠEK, Jindřích. RADOVÁ, Vlasta. 4.3 Zpracování v časové oblasti. *Mluvíme s počítačem česky*. 1 vydání. Praha: nakladatelství Academia, 2006, 738 stran. ISBN 80-200-1309-1.

[2.5] Asymptotická složitost. w*ww.algoritmy.net* [online]. [cit. 06.03.2018]. Dostupné na WWW: https://www.algoritmy.net/article/102/Asymptoticka-slozitost

[2.6] Cooley-Turkey FFT algorithm. *en.wikipedia.com* [online]. [cit. 6.3.2018]. Dostupné na WWW: https://upload.wikimedia.org/wikipedia/commons/c/cb/DIT-FFT-butterfly.png

[2.7] VEDALA, Krishna. *cs.wikipedia.org* [online]. [cit. 27.2.2018]. Dostupné na WWW: https://commons.wikimedia.org/wiki/File:Mel-Hz_plot.svg

[2.8] Practical Cryptography. *Practical Cryptography* [online]. Copyright © 2009 [cit. 03.04.2018]. Dostupné na WWW: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

[2.9] PSUTKA, Josef. MÜLLER, Luděk. MATOUŠEK, Jindřích. RADOVÁ, Vlasta. 4.6.4 Melovské kepstrální koeficienty. *Mluvíme s počítačem česky*. 1 vydání. Praha: nakladatelství Academia, 2006, 738 stran. ISBN 80-200-1309-1.

[2.10] DTW warping path. Dynamic Time Warping Techniques for Missing Value. [online] cst.tu-plovdiv.bg [cit. 27.2.2018]. dostupné na WWW: http://cst.tu-plovdiv.bg/bi/DTWimpute/DTWalgorithm.html

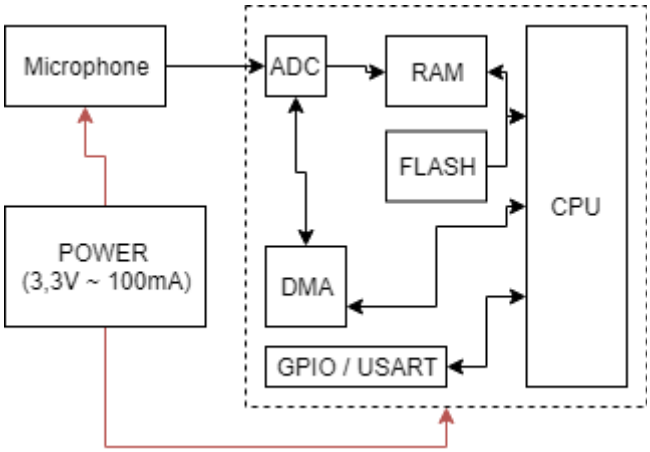
3. Hardware

Pro záznam zvukového signálu je zapotřebí hardware, který bude schopen citlivě zaznamenat audio signál. Pro aplikaci na hlasové příkazy stačí zapojení jednoduchého mikrofonu se zesílením pomocí operačního zesilovače, který bude mít velké zesílení pro pokrytí co největšího pásma rozsahu AD převodníku. Vzhledem k tomu, že výstup z mikrofonu vzorkujeme pomocí AD převodníku s frekvencí $f_{sample} = X[Hz]$, je potřeba omezit maximální výstupní frekvenci mikrofonu na $X_{output}[Hz] < f_{sample}$ z důvodu aliasing efektu.

Frekvenční rozsah přenášeného mluveného slova zajišťující srozumitelnost se pohybuje v oblasti 300Hz - 3400 Hz [3.1]. Vyšší frekvence se podílí hlavně na barvě hlasu, což je pro náš účel zanedbatelné. Omezení frekvence je realizovatelné pomocí relativně jednoduchého zapojení dolnopropustního filtru (dále jen LP popřípadě LPF[4]). Ten je realizovaný pomocí RC článku v zapojení jako integrační člen.

Bylo by vhodné omezit i frekvence na nízkých frekvenčních hladinách. Toto omezení je výhodné hlavně z důvodu filtrace frekvence napájecích zdrojů. Nicméně cílený hardware je napájen stejnosměrným napětím a stabilizován interním obvodem. Většina mikrofonů má svůj frekvenční rozsah omezený konstrukcí, takže není nutné nižší frekvence příliš omezovat. Běžný frekvenční rozsah standartního mikrofonu se pohybuje v rozmezí 20Hz – 16kHz.

Mikroprocesor má velmi omezený výpočetní výkon. Snahou je omezit výpočetní náročnost. Většinu softwarových operací lze předejít i velmi jednoduchým zapojením v hardwarové části. Výše uvedené dolnopropustní filtry na úrovni hardwaru mohou značně snížit výpočetní a paměťovou náročnost výsledného softwaru.



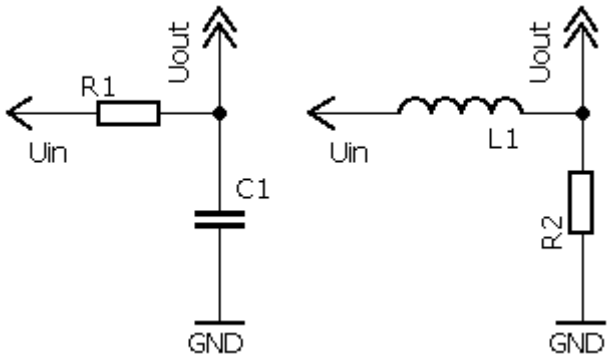
Obrázek 3.1: Blokové schéma zapojení

4 LPF – low pass filter = běžné označení zkratkou v anglickém jazyce pro dolnopropustný filtr

Mikroprocesor (dále jen MCU[5]) na kterém byl proveden celý vývoj, je 32-bitový procesor od firmy ATMEL [3.1]. Jedná se o typ AT32UC3L064. Pro usnadnění vývoje se využil vývojový kit obsahující tento procesor UC3-L0 XPLAINED [3.2]. Kit disponuje velkým množstvím zapojení pro experimenty a počáteční vývoj bez nutnosti komplikovaného návrhu zapojení a výroby vlastních plošných spojů. Velmi jednoduché blokové zapojení je na obrázku 3.1.

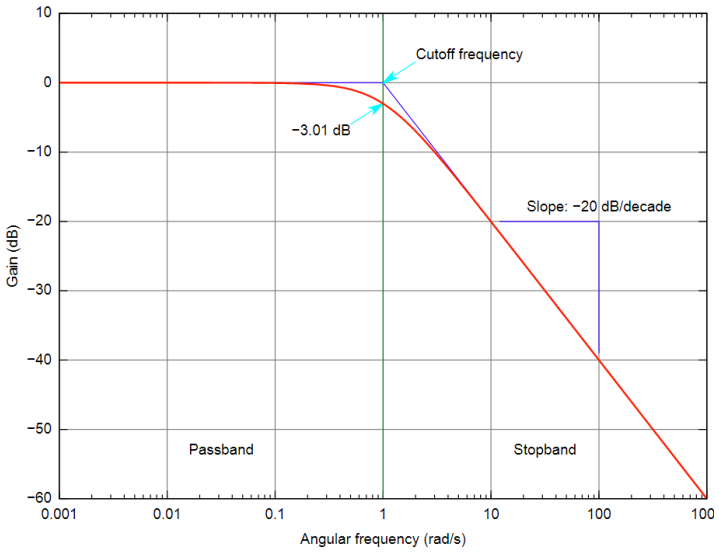
3.1 Dolnopropustný frekvenční filtr

Integrační článek nebo-li dolnopropustný filtr propouští nižší frekvence, než je tkz. cutoff frekvence s útlumem 20dB/dek. Nejjednodušší integrační článek lze získat zapojením RC nebo RL obvodu.



Obrázek 3.2: RC a RL LP filtry

Na předchozím obrázku jsou znázorněny pasivní filtry typu dolní propusti. U_{in} je vstupní napětí s maximální frekvencí, kterou chceme utlumit. U_{out} je výstupní napětí s útlumem požadovaných frekvencí. V našem případě je využito zapojení RC.

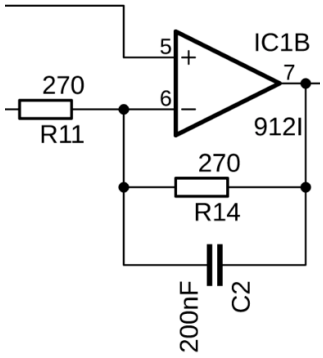


Graf 3.1: LAFCH LP filtru s cutoff frekvencí na 1Hz. [3.3]

5 MCU – microcontroller unit, neboli jednočipový počítač

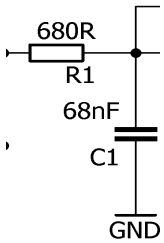
LPF je navržen pomocí RC obvodu za použití součástek o hodnotách R=270Ω a C=200nF. Dle vzorce lze vypočítat cutoff frekvenci: $f_{cutoff} = \frac{1}{2\pi RC} = \frac{1}{2\pi * 270 * 200\text{e-}9} = 2947\text{Hz}$ [3.4]. Vzorkovací frekvence AD převodníku MCU je nastavena na 8kHz. Bohužel útlum by mohl být příliš malý. Výstup by neměl obsahovat žádné složky s frekvenci 4kHz a více.

Z tohoto důvodu je v zapojení osazen ještě jeden LPF s lehce vyšší cutoff frekvencí. $f_{cutoff} = \frac{1}{2\pi RC} = \frac{1}{2\pi * 680 * 68 - 9} = 3441\text{Hz}$ Tento filtr má parametry součástek R=680Ω a C=68nF. Uvedené hodnoty cutoff frekvencí jsou v ideálním případě.



Obrázek 3.3: První stupeň LP filtru

Na předchozím obrázku je schéma prvního stupně LPF. Tento filtr je zapojen jako aktivní filtr pomocí operačního zesilovače. Na pin 5 je přivedeno poloviční napájecí napětí $\frac{U_{cc}}{2}$ z důvodu posunutí "virtuální" nuly. Ve zpětné vazbě je poté zapojen integrační článek a záporná zpětná vazba s jednotkovým záporným zesílením. Pro tento daný případ se cutoff frekvence vypočte pro hodnoty součástek R=R_{14} a C=C_2.Výstup OZ (operační zesilovač) je poté přiveden na druhý stupeň LPF v provedení jednoduchého pasivního RC integračního článku.



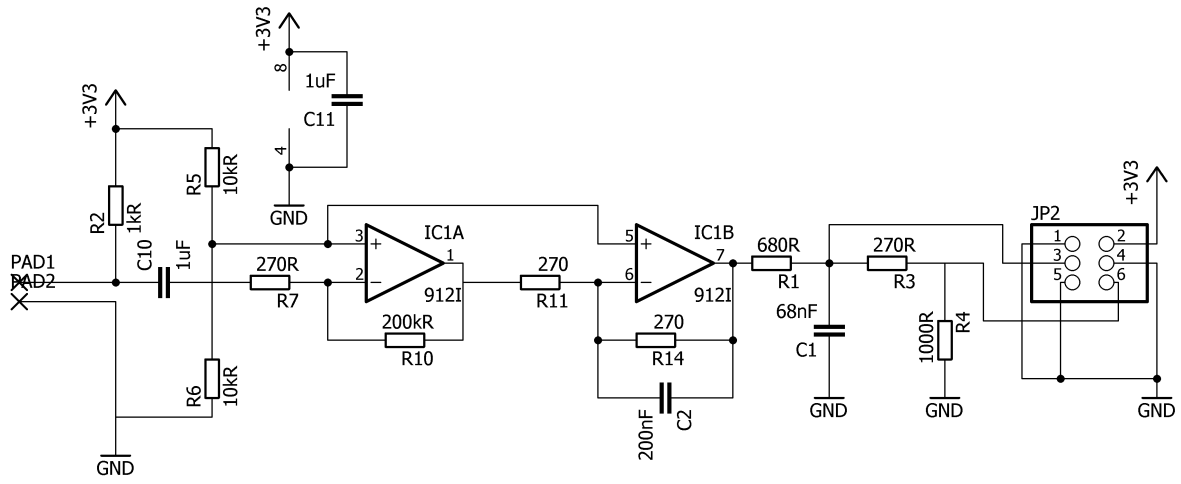
Obrázek 3.4: Jednoduchý LP filtr druhý stupeň

Předchozí obrázek znázorňuje jednoduchý LPF. Pro tento případ je výpočet cutoff frekvence pro součástky R=R_1 a C=C_1.

3.2 Zapojení mikrofonu

Pro mikrofon je zvolena elektretová mikrofonní vložka. Jedná se o velmi běžný mikrofon fungující na principu proměnlivé kapacity. Tyto mikrofony mají rozdílné frekvenční charakteristiky lišící se dle konstrukce.

Použitá mikrofonní vložka je typu MCE-100 [3.5]. Tento mikrofon má definovaný frekvenční rozsah 50Hz – 10kHz. Výstupní impedanci 2,2kΩ a rozsah napájení 1,5V – 10V. Výstup je připojen v sérii s vazebním kondenzátorem na operační zesilovač.



Obrázek 3.5: Schéma zapojení mikrofonu

Předchozí obrázek je schéma zapojení mikrofonu. Mikrofonní vložka je připojena na pinech PAD1 a PAD2 (PAD2 je uzemněný, tedy na tento pin se musí připojit zem mikrofonu). R_2 je předřadný rezistor a vazební kondenzátor C_{10} slouží k propusti proměnlivé složky napětí (blokuje stejnosměrné napětí – zdroj napětí).

Rezistory R_5 a R_6 slouží jako dělič napětí pro posunutí pracovního napětí ("virtuální" nula). Za předpokladu napájecího napětí U_{cc} = 3,3V a R_5=R_6=10kΩ se bude výstupní napětí této napěťové děličky pohybovat okolo 1,65V.

Jako OZ je použit rail-to-rail OZ typu 912I. Rail-to-rail je použit z důvodu minimálního "ořiznutí" výstupního napětí. Kdyby nebyl použit rail-to-rail, tak při takto nizkém napětí by mohlo dojít k omezení výstupního napětí až o 1,5V (dle použitého OZ) shora i spoda.

Zapojení IC1A je v zapojení invertující zesilovač se zesílením $A = \frac{-R_{10}}{R_7} \approx -740$.

Vzhledem k tomu, že vstup do IC1A je velmi malý (řádově mV), tak je potřeba výstup značně zesílit. Bylo by výhodné zesílení rozdělit na dva stupně pro omezení šumu vzniklého při vysokém zesílení. Tento šum je dán neideálním operačním zesilovačem (vliv napěťového a proudového

offsetu). Například IC1A se zesílením -20 a IC2B se zesílením -40. Celkové zesílení by bylo v součinu 800. Nicméně IC1B je cíleně limitováno pouze jako dolnopropustný filtr. Složitější návrh je vynechaný, pro zjednodušení a zmenšení počtu potřebných součástek.

Zapojení IC1B je zapojené jako aktivní LPF s -1 zesílením viz. kapitola 3.1. Na výstupu IC1B je připojen pasivní LPF a napěťová dělička. Výstupní napětí na vstup AD převodníku MCU musí být omezeno na maximální hodnotu rovnající se referenčnímu napětí AD převodníku. Referenční napětí je 1,8V. Které je získáno z MCU interním regulátorem napětí.

Zapojení rezistorů R_1, R_3 a R_4 funguje jako napěťová dělička, která "přenásobí" výstupní napětí hodnotou menší než 1. Pro ověření, že výstupní napětí U_o nepřesáhne povolené napětí AD převodníku MCU, se vypočte jednoduchým vztahem:

$$U_o = \frac{R_4}{R_4 + R_1 + R_3} U_I = \frac{1000}{1000 + 680 + 270} = \frac{1000}{1950} * 3{,}3 = 1{,}692\,V$$

Výstupní napětí, za předpokladu pevného napájecího napětí o velikosti 3,3V a ideálních součástek, nemůže přesáhnout napětí 1,692V. Vzniká zde problém se sníženou přesností. Za předpokladu 12-bitového AD převodníku (to znamená 2^{12} = 4096 hodnot) využijeme pouze 94,02% rozsahu, dle vztahu $W[\%] = \frac{U_{max\,act}}{U_{ref}} * 100, U_{ref}$ *je referenční napětí AD převodníku* . To znamená, že výstup ideálního AD převodníku bude v rozmezí 0-3850 dle vztahu $ADC_{range} = W * 4096 = 0{,}940 * 4096 = 3850$ nebo lépe $ADC_{range} = \frac{U_{max} * 2^M}{U_{ref}} = \frac{1{,}692 * 2^{12}}{1{,}8}$.

Bohužel toto omezení nelze zlepšit z důvodu limitace dostupných součástek (vyrábí se pouze některé hodnoty). Pokud změníme některé hodnoty ať už kondenzátorů nebo odporů, je pak vhodné znovu natrénovat vzorové příkazy do MCU, protože dojde k ovlivnění vstupu jako celku.

JP2 je konektor pro připojení mikrofonu k cílovému zařízení a C_{11} je kapacitor sloužící k vyrovnání případného rušení napájecího zdroje. Celý modul je poté možné připojit pomocí alespoň třívodičového kabelu (ideálně stíněný kabel nebo kroucená dvojlinka pro minimalizaci rušení externími zdroji).

3.3 Mikroprocesor AT32UC3L064

Mikroprocesor nebo-li mikrokontrolér je součástka většinou integrující velké množství periférií. Mezi periférie se řadí binární vstupy a výstupy (I/O), časovače, AD převodníky, paměť RAM a nebo i EEPROM, atd. Jedná se tedy o procesor schopný výpočtů, komunikace, měření či

vyhodnocení dat na svých vstupech. Vše je závislé na naprogramování a typu procesoru.

Mikroprocesor, na kterém probíhal vývoj je AT32UC3L064 (dále jen UC3). Jedná se o 32bit procesor s maximální výpočetní rychlostí 64 DMIPS, pamětí FLASH o velikosti 64KB[6] a RAM o velikosti 16KB. Pro naši potřebu disponuje AD převodníkem s maximálním rozlišením 12-bit, několika časovači, DMA[7] a komunikačním rozhraním USART nebo TWI (kompatibilní s I2C).

Feature	AT32UC3L064	AT32UC3L032	AT32UC3L016
Flash	64KB	32KB	16KB
SRAM	16KB	16KB	8KB
GPIO	36		
High-drive pins	5		
External Interrupts	6		
TWI	2		
USART	4		
Peripheral DMA Channels	12		
Peripheral Event System	1		
SPI	1		
Asynchronous Timers	1		
Timer/Counter Channels	6		
PWM channels	36		
Frequency Meter	1		
Watchdog Timer	1		
Power Manager	1		
Secure Access Unit	1		
Glue Logic Controller	1		
Oscillators	Digital Frequency Locked Loop 40-150MHz (DFLL) Crystal Oscillator 3-16MHz (OSC0) Crystal Oscillator 32KHz (OSC32K) RC Oscillator 120MHz (RC120M) RC Oscillator 115kHz (RCSYS) RC Oscillator 32kHz (RC32K)		
ADC	8-channel 12-bit		
Temperature Sensor	1		
Analog Comparators	8		
Capacitive Touch Module	1		
JTAG	1		
aWire	1		
Max Frequency	50MHz		
Packages	TQFP48/QFN48/TLLGA48		

Tabulka 3.1: Souhrn dostupných periférií mikroprocesoru [3.6]

Celá vývojová deska je napájena z USB portu. Vstupní napětí je 5V. Toto napětí je poté stabilizováno napěťovým regulátorem na velikost 3,3V pro napájení mikroprocesoru. Mikroprocesor je zapojen tak, že vstupní a výstupní napětí I/O je 3,3V. Podle výrobce referenční napětí AD převodníku je připojeno na interní napěťový regulátor 1,8V. Pokud je pin MCU využit jako vstup AD převodníku, tak na tento pin by nemělo být přivedeno napětí vyšší než 1,8V.

6 KB – zkratka význam: kilobajt – 1000 bajtů
7 DMA – direct memory access – přímý přístup do paměti. Slouží k přenosu informací bez účasti procesoru.

Informace o této problematice viz. kapitola 3.2.

UC3 má několik dostupných zdrojů pro takt jádra procesoru s omezením na maximální frekvenci procesoru 50MHz. Vzhledem k celkové výpočetní náročnosti celé aplikace je vhodné procesor taktovat na co nejvyšší frekvenci, tj. 50MHz. Využitím externího hodinového krystalu[8] s DFLL[9] je procesor taktován přibližně na frekvenci 49971200Hz. Tato frekvence je výsledkem celočíselného zaokrouhlování. Více specifikací a informací ohledně vývojové desky je na stránkách výrobce. Nevýhodou zvýšení frekvence pomocí DFLL je, že v případě odchylky nebo nestability zdrojové frekvence, může dojít k překročení maximálně přípustné hodnoty. V tomto případě by mohlo dojít k nestabilitě procesoru jako celku nebo dokonce k nevratnému poškození! Tento fakt je nutné brát na zřetel a proto je vhodné ověřit maximálně přípustnou hranici. Hodinový krystal má standardně 32768Hz +/- 80ppm[10]. Maximální frekvence je 32770Hz a minimální frekvence 32765Hz. Velikost násobení pomocí DFLL je v našem případě 1525x, proto nejvyšší frekvence dosahuje 49974250Hz, což je ještě v bezpečné normě.

AD převodník procesoru lze nastavit na periodické vyhodnocování analogového vstupu. Tato možnost nastavení je výhodná pro vzorkování audio signálu a ve spojení s PDCA (interní název pro DMA využívající převod dat mezi periferiemi), lze dosáhnout vzorkování s minimální účastí procesoru. Procesor se tedy věnuje výpočtům bez častého přerušování vyvolané periferiemi, což je často náročné na prostředky. DMA čte hodnotu na výstupu AD převodníku po skončení převodu a uloží výsledek na určenou pozici v paměti RAM. Periodické vyhodnocování je nastaveno pomocí interního časovače samotného AD převodníku. Frekvence AD převodníku je nastavena na hodnotu 3123200Hz (maximální uváděná frekvence je 6MHz). Tato hodnota je dosažena dělením taktovací frekvence jádra děličkou o velikosti 16. Dle vzorce $f_{convert} = \frac{1}{AD_{period}} * f_{ADclk}$ můžeme vypočítat potřebnou konstantu AD_{period} pro nastavení časovače pro periodické vyhodnocování. $AD_{period} = \frac{1}{f_{convert}} * f_{ADclk} \rightarrow AD_{period} = \frac{1}{8192} * 3123200 = 381$. Kde $f_{convert}$ je požadovaná vzorkovací frekvence (bylo zvoleno 8kHz přesněji 8192Hz) a f_{ADclk} je taktovací frekvence AD převodníku.

8 Hodinový krystal je běžně používaný název pro krystal s frekvencí 32768Hz = 2^{15}Hz

9 DFLL – digital frequency locked loop. Jedná se o "násobičku" frekvence.

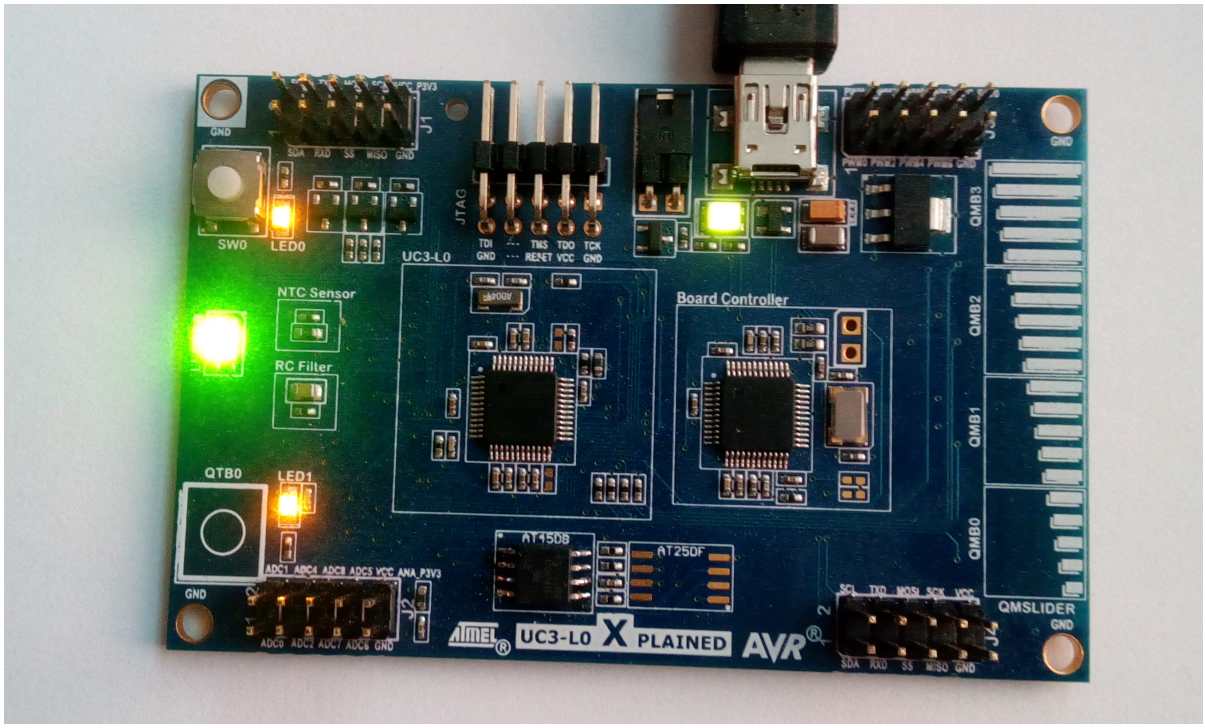
10 PPM – parts per million. V případě krystalu udávaná "chybovost" v milionech. Například 10 000 PPM znamená chybovost 1%.

Taktovací frekvence	Požadovaná hodnota	Minimální hodnota	Maximální hodnota
Jádro procesoru	49971200Hz	49966625Hz	49974250Hz
AD převodník	3123200Hz	3122914Hz	3123390Hz
Vzorkovací frekvence AD	8197Hz	8196Hz	8197Hz

Tabulka 3.2: rozsah hodnot frekvencí v závislosti na přesnosti krystalu

Předchozí tabulka znázorňuje meze frekvencí procesoru, což ovlivní frekvenci AD převodníku a vzorkovací frekvenci. Z důsledku celočíselného zaokrouhlování je výsledná vzorkovací frekvence 8196,5Hz +/- 0,5Hz.

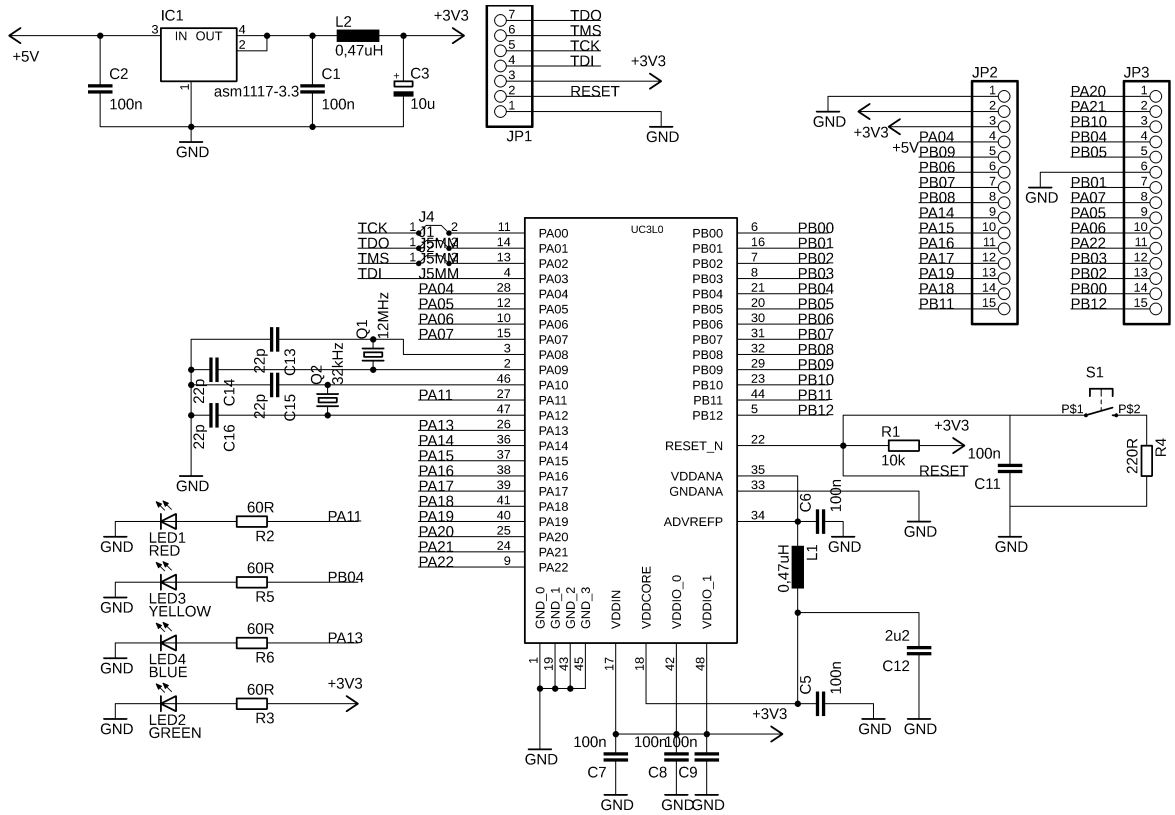
Na následujícím obrázku je reálná fotografie vývojové desky. Uprostřed plošného spoje se nachází mikroprocesor UC3. V pravé části je pomocný mikroprocesor zajišťující komunikaci s připojeným počítačem a programování mikroprocesoru UC3 i bez programátoru. Plošný spoj obsahuje 5 konektorů. Konektor JTAG pro připojení stejnojmenného rozhraní. Poté 4 konektory v podobě dvouřadých kolíkových lišt. Každý z těchto konektorů má jistou oblast využití: J1 a J4 (vlevo nahoře a vpravo dole) má vyvedeny piny pro použití komunikace (USART, SPI, TWI). J2 (vlevo dole) má vyvedeny piny pro využití AD převodníku. J3 (vpravo nahoře) má piny vyvedeny pro využití výstupů časovačů (většinou využití pro PWM). Celá destička má plošné rozměry 85x54mm.



Obrázek 3.6: UC3-L0 Xplained

Je viditelné, že deska vývojového kitu obsahuje příliš mnoho obvodů, které nemají funkční význam pro cílenou aplikaci, ale jsou výhodné pro experimenty.

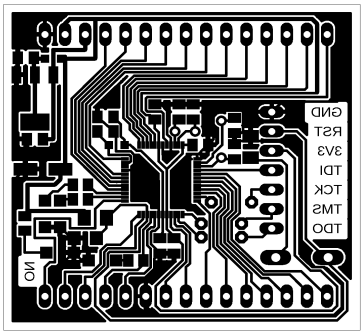
Byl vytvořen návrh obvodu se stejným mikroprocesorem se základním zapojením pro jeho funkci s volnými piny (kromě zapojených pár LED diod), aby tyto piny zůstaly volné pro účely i jiných projektů. V základě je zapojení jednoduché. Mikroprocesor je napájen stabilizátorem o napětí 3,3V. Vnitřní stabilizátor napětí 1,8V je využit pro napájení jádra procesoru a AD převodníku. Jedná se tedy o stejný typ zapojení jako na vývojové desce. Mikroprocesor má možnost připojení hodinového krystalu a externího krystalu, proto jsou obě dvě varianty připravené k osazení. Rozteč kolíkových lišt na které jsou vyvedeny piny procesoru je taková, aby se dala destička zapojit do širšího nepájivého pole.



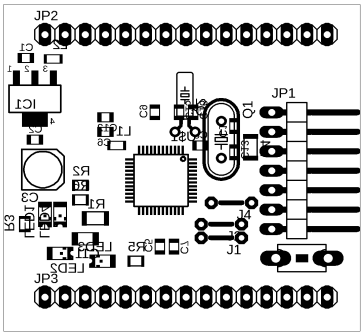
Obrázek 3.7: Schéma zapojení

Bylo využito hlavně SMD součástek s minimální velikostí 0805 (2mm x 1,27mm). Existuje možnost výsledný obvod ještě zmenšit, ale účel bylo vytvořit obvod, který by bylo možné vyrobit v amatérských podmínkách. To znamená, vyloučení příliš malých součástek a součástek, které je obtížné sehnat. Na obrázku 3.7 je schéma zapojení mikroprocesoru. Jedná se o co nejjednodušší zapojení bez zvláštních obvodů. Konektory JP2 a JP3 slouží k vývodům signálů z procesoru. Konektor JP1 je JTAG rozhraní s nestandartním rozložením vývodů. Vazební kondenzátory C13, C14, C15 a C16 se u krystalů mohou lišit oproti návrhu. Stejně tak hodnota krystalu Q1. Tlačítko S1 slouží k hardwarovému resetu procesoru.

Na obrázku 3.8 je deska plošných spojů. Je nutné podotknout, že zamýšlená orientace je vzhůru nohama. Tedy při používání bude mikroprocesor natočen nahoru. Jedná se o jednostrannou desku s rozměry 43x45mm. Cíl bylo navrhnout co nejmenší funkční a jednoduchou destičku s možností univerzálního zapojení do jiného projektu. Konektory JP2 a JP3 tedy slouží podobně jako patice u desktopových procesorů. Navržením "základní" desky je poté možné tento procesor připojit bez nutnosti pájení či jiných úprav na desce procesoru.



Obrázek 3.8: deska plošných spojů



Obrázek 3.9: Rozložení součástek

Literatura

[3.1] Wideband audio - Wikipedia. [online]. [cit. 22.03.2018]
Dostupné na WWW: https://en.wikipedia.org/wiki/Wideband_audio

[3.2] General information - - UC3-L0 Xplained. *Home* | *Microchip Technology Inc.*[online]. [cit. 20.02.2018]
Dostupné na: https://www.microchip.com/webdoc/uc3l0xplained/uc3l0xplained.chapter.poj_mfp_kc.html

[3.3] Response first-order low-pass filter. Wikipedia. [online].[cit. 22.03.2018]
Dostupné na: https://en.wikipedia.org/wiki/Low-pass_filter

[3.4] Passive Low Pass Filter.. ElectronicsTutorials. [online].
[cit. 22.03.2018]. Dostupné na: https://www.electronics-tutorials.ws/filter/filter_2.html

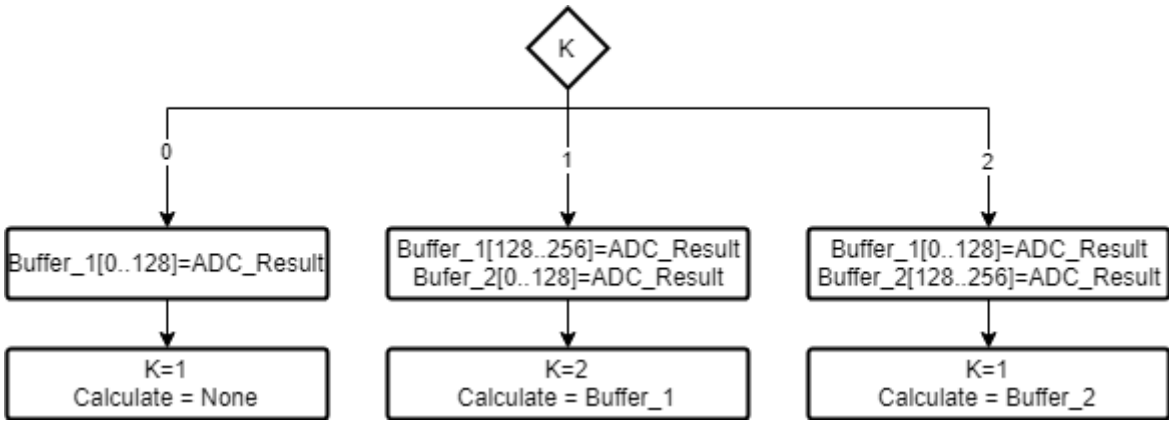
[3.5] MCE100 NEW2 | GM electronic, spol. s.r.o.. *GM electronic* | *elektronické součástky, komponenty .* | *GM electronic, spol. s.r.o.* [online]. [cit. 20.02.2018] Dostupné na: https://www.gme.cz/mce100-new2

[3.6] Atmel Corporation 2012. Tabulka 2-1 str. 6. [online]. [cit. 22.03.2018]
Dostupné na: http://ww1.microchip.com/downloads/en/DeviceDoc/doc32099.pdf

4. Software

Program pro mikroprocesor je psaný v jazyce C. Pro vývoj programů na platformu Atmel je přímo od výrobce distribuované a podporované vývojové prostředí AtmelStudio [4.1]. AtmelStudio zajišťuje překlad do strojového kódu a pro optimalizaci kódu, jestliže je zapnutá (a dle zvolené úrovně optimalizace). Tato optimalizace je užitečná, protože upravuje program. Například vkládáním často volaných funkcí v cyklech, nebo přepis volání polí na pointery. Součástí aplikace AtmelStudio jsou i různé knihovny optimalizované pro zvolené procesory, usnadňující vývoj. Tyto knihovny obsahují funkce zajišťující správné nastavení například frekvence procesoru, operace s GPIO, ADC nastavení atd. Výstupní program je ve formátu intel hex. Tento formát je často používaný pro běžné dostupné programátory mikroprocesorů.

Diagram 4.1 zobrazuje zjednodušenou podobu programu. Při každém přerušení DMA dojde k výpočtu MFC koeficientů a uložení do paměti MCU. Tato paměť historie je omezena na velikost 128 příznakových vektorů. V případě 12ti příznaků je vyhrazená velikost v paměti pro toto datové pole 128*12 Bajtů (1536 Bajtů). V okamžiku, kdy energie analyzovaného mikrosegmentu překročí práh, spustí se rozpoznávání. Práh je vypočten jako průměr energie deseti posledních okének. V okamžiku, kdy je proměnná rozpoznávání = True, tak vždy když MCU nemusí analyzovat další okénko, stráví veškerý procesorový čas výpočtem DTW pro jednotlivé nahrané vzory.



Obrázek 4.1: Zajištění překryvu okének

Délka příkazu je z důvodu omezené historie maximálně 2 sekundy. V případě, kdy energie mikrosegmentu překročí práh, zvolí se jako počátek rozpoznávání okénko nacházející se na indexu s hodnotou o 20 méně, než je aktuální pozice (hodnota 20 zvolena experimentálně). Program běží od okamžiku, kdy je MCU připojeno na zdroj napájení. popřípadě v momentě, kdy se navrátí z resetu. Terminace běhu programu dojde v momentě odpojení od zdroje napájení nebo v případě softwarového přerušení (dedikované příkazy), popřípadě při resetu. V diagramu není zobrazené řešení překryvu okének. To je dosaženo dvěma buffery hodnot pro výsledky z AD převodníku.

PDCA je nastaveno na přerušení pro každých 128 vzorků ADC. V lichém přerušení se výsledek nakopíruje do první poloviny bufferu_1 a do druhé poloviny bufferu_2. V sudém přerušení se výsledek nakopíruje do druhé poloviny bufferu_1 a do první poloviny buferu_2. Tím, že je plnění bufferu odsazeno o jedno okénko dojde v překryvu 50% a analýze nezávisle na sobě. Znázornění je na diagramu 4.2. Vzhledem k velmi omezeným prostředkům je často zvolena velmi úsporná metoda pro ukládání dat do paměti RAM. Tyto metody zhoršují čitelnost kódu a samotný vývoj, ale efektivně sníží paměťovu náročnost za velmi malý úkor výpočetního vytížení.

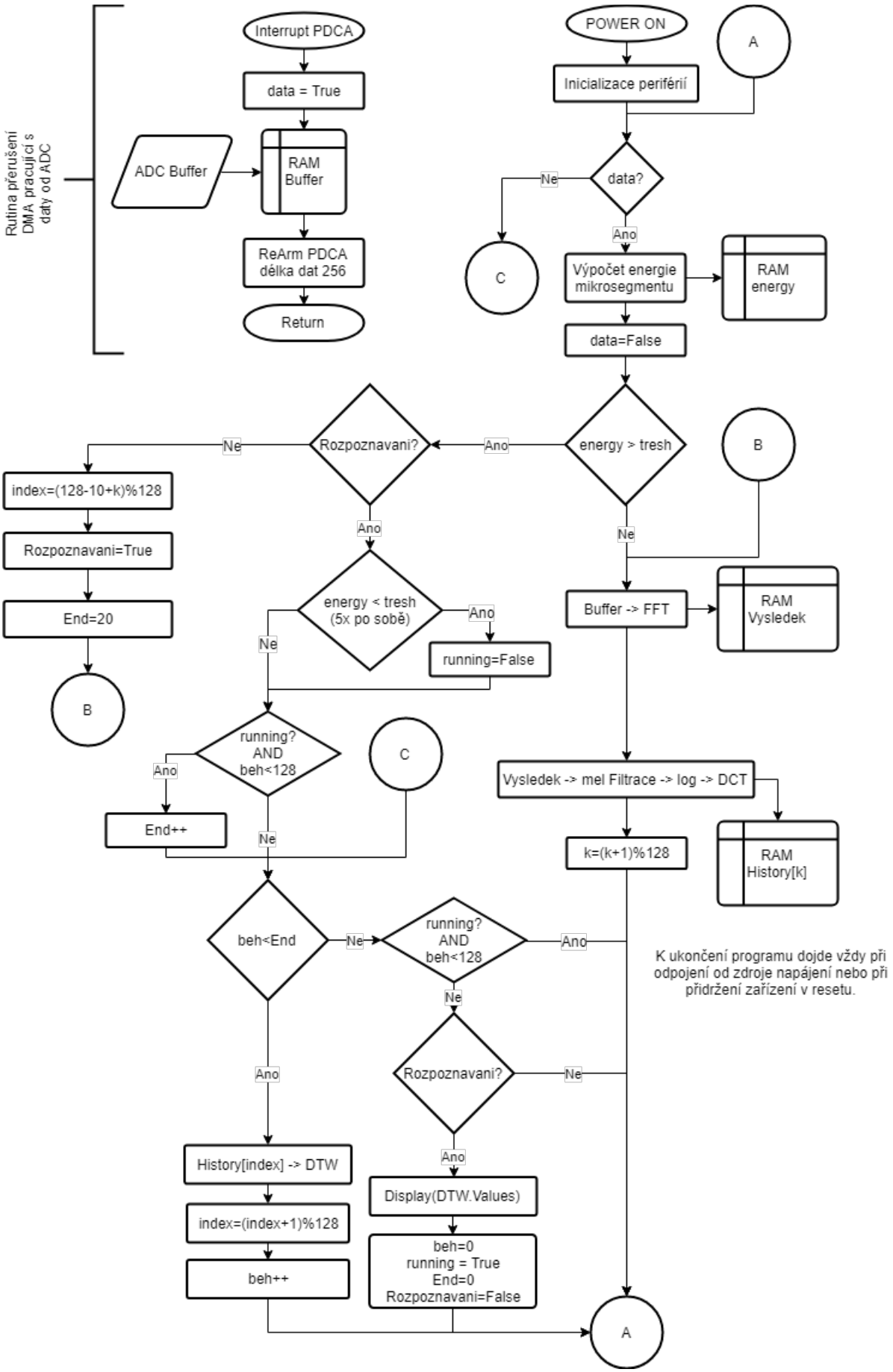


Diagram 4.1: Zjednodušený diagram programu

4.1 Inicializace MCU

Při zapnutí mikroprocesoru, respektive při připojení napájení, se mikroprocesor řídí frekvencí z interního oscilátoru. Tento oscilátor má mnohem nižší frekvenci, než je maximálně povolená výrobcem pro takt MCU a to 115kHz. Tento oscilátor funguje prioritně jako výchozí zdroj taktovací frekvence pro účelné nastavení a kontrolu stabilizace nebo jako záložní zdroj, pokud dojde k selhání vybraného zdroje taktovací frekvence (dále jen F_{source}).

```
sysclk_init();      //Nastavení zdroje taktovací frekvence
gpio_clr_gpio_pin(LED0_GPIO);     //▼▼▼ Vypnutí LED diod ▼▼▼
gpio_clr_gpio_pin(LED1_GPIO);
gpio_clr_gpio_pin(LED2_GPIO);
gpio_clr_gpio_pin(LED3_GPIO);
gpio_clr_gpio_pin(LED4_GPIO);     //▲▲▲ ********** ▲▲▲
init_USART1();      //Inicializace USART
init_ADC();         //Inicializace ADC
init_PDCA();  //Inicializace DMA
```

Blok 4.1: Inicializační segment

Využitím dostupných funkcí v knihovnách je nastavení požadované F_{source} relativně jednoduché. V bloku 4.1 je výřez kódu zajišťující volání jednotlivých inicializací v sekci main(). Funkce sysclk_init() poté pomocí definovaných hodnot přenastaví potřebné registry MCU pro aktivaci DFLL a vyčkání ve smyčce, než dojde ke stabilizaci (stabilizaci kontroluje interní registr MCU). V bloku jsou poté volané inicializační funkce pro USART (sériová linka), ADC (AD převodník) a PDCA (DMA).

```
void init_USART1(void){
	usart_options_t usart_opt = {
		.baudrate     = 921600,      //baud rychlost
		.channelmode  = USART_NORMAL_CHMODE,      //std. mod komunikace
		.charlength   = 8,    //delka znaku
		.paritytype   = USART_NO_PARITY,    //bez parity
		.stopbits     = USART_1_STOPBIT,    //1 stop bit
	};
	//urceni portu pro USART
	const gpio_map_t usart_gpio_map = {
		{USART0_RXD_PIN, USART0_RXD_FUNCTION},
		{USART0_TXD_PIN, USART0_TXD_FUNCTION}
	};
	//aktivace modulu
	gpio_enable_module(usart_gpio_map,
	sizeof(usart_gpio_map) / sizeof(usart_gpio_map[0]));
	//zapnuti frekvence pro USART
	sysclk_enable_peripheral_clock(USART0);
	usart_init_rs232(USART0, &usart_opt,
	sysclk_get_peripheral_bus_hz(USART0));
}
```

Blok 4.2: Inicializační rutina pro USART

Blok 4.2 je úsek programu funkce pro inicializaci USART linky MCU. Ostatní inicializační funkce periférií vypadají vesměs stejně. Nachází se zde datová struktura obsahující hodnoty potřebné pro nastavení periférie (`usart_options_t`) a datová struktura obsahující hodnoty potřebné pro nastavení pinů (gpio_map_t) pro funkci dané periférie. Funkce gpio_enable_module() je pro aktivaci daných pinů nastavené strukturou, sysclk_enable_peripheral_clock() je pro zapnutí taktovací frekvence pro periférii. Funkce usart_init_rs232() zajišťuje správné nastavení registrů USART, pomocí funkce z knihovny. Téměř každá periférie, která pro svoji činnost potřebuje nějaký pin MCU potřebuje tento pin určit. Jeden pin může být využit několika periférie mi, ale nikdy současně. Proto by se měly tyto piny určit při startu programu, popřípadě zajistit neblokující se přístup.

```
uint64_t temp0;
uint64_t temp1;
uint64_t tempX;
int32_t maximum=0;
for(k=0; k<okenkoHalf; k++){
	temp0=abs(ADCBuffer2R[k]);
	temp1=abs(ADCBuffer2I[k]);
	tempX=temp0*temp0;
	tempX+=temp1*temp1;
	vysledek[k]=isqrt(tempX);
	if(vysledek[k]>noise[k])
		vysledek[k]-=noise[k];
	else
		vysledek[k]=0;
}
```

```
uint32_t isqrt(uint64_t x)
{
	if (x == 0 || x == 1)
		return x;
	uint64_t start=1, endX=x, ansX, mid, midX;
	while (start <= endX)
	{
		mid = (start + endX)/2;
		midX=mid*mid;
		if (midX == x)
			return mid;
		if (midX < x)
		{
			start = mid + 1;
			ansX = mid;
		}
		else
			endX = mid - 1;
	}
	return ansX;
}
```

Blok 4.3: Výpočet amplitud jednotlivých frekvencí

4.2 Výpočet FFT

Výpočet FFT je náročný na hardwarové prostředky. Částečně lze tuto zátěž snížit pomocí předvypočtených hodnot v lookup tabulce. Hodnoty jako sin a cos nebo tabulka zajišťující zpřeházení vstupních dat se tedy budou považovat za konstanty na základě indexu. Ve výpočtech se vůbec nevyskytují hodnoty ve formátu plovoucí čárky (float, double). Práce s plovoucí čárkou je výpočetně velmi náročná, proto je nahrazena operacemi s fixní čárkou. Fixní čárka oproti plovoucí čárce má horší přesnost, ale je podstatně méně náročná pro výpočty, neboť se s ní pracuje jako se standartním celým číslem (int, short, …).

```
	static uint32_t dokonceniMel=0;
	uint32_t np=1,poradi=1,n2=2,k,n1=1,j=0;
	int32_t t1,t2;
	int16_t c=4096,s=0;
	for(k=0; k<okenko; k++){
		ADCBuffer2R[k]=data[reverse[k]];  //naplnění daty
		ADCBuffer2I[k]=0;    //anulace bufferu
	}
	for (k=0; k<okenko; k+=n2,np+=n2){       //prvni faze vypoctu
		t1 = ((c*ADCBuffer2R[np]))>>12;
		ADCBuffer2R[np] = ADCBuffer2R[k] - t1;
		ADCBuffer2R[k] += t1;
		ADCBuffer2I[k] = (s*ADCBuffer2R[np])>>12;
		ADCBuffer2I[np] = -ADCBuffer2I[k];
	}
	n1 = n2;
	n2<<=1;
	for(j=0; j < n1; j++){     //druha faze vypoctu
		c=cosT[poradi];
		s=sinT[poradi++];
		np=n1+j;
		for (k=j; k < okenko; k+=n2,np+=n2)
		{
			t1 = ((c*ADCBuffer2R[np]))>>12;
			t2 = ((s*ADCBuffer2R[np]))>>12;
			ADCBuffer2R[np] = ADCBuffer2R[k] - t1;
			ADCBuffer2I[np] = -t2;
			ADCBuffer2R[k] += t1;
			ADCBuffer2I[k] = t2;
		}
	}
	for(unsigned char i=2; i < mocnina; i++){       //treti faze vypoctu
		n1 = n2;
		n2<<=1;
		for(j=0; j < n1; j++){
			c=cosT[poradi];
			s=sinT[poradi++];
			np=n1+j;
			for (k=j; k < okenko; k+=n2,np+=n2)
			{
				t1=((c*ADCBuffer2R[np])-(s*ADCBuffer2I[np]))>>12;
				t2=((s*ADCBuffer2R[np])+(c*ADCBuffer2I[np]))>>12;
				ADCBuffer2R[np] = ADCBuffer2R[k] - t1;
				ADCBuffer2I[np] = ADCBuffer2I[k] - t2;
				ADCBuffer2R[k] += t1;
				ADCBuffer2I[k] += t2;
			}
		}
	}
```

Blok 4.4: Výpočet FFT

Blok 4.3 je ukázka výpočtu amplitud FFT. Z důvodu možných velmi vysokých výsledných hodnot energie jsou hodnoty dočasně uloženy do proměnných o velikosti 64bit. Proměnná tempX je poté odmocněna pomocí estimované verze odmocniny. Funkce isqrt najde nejbližší celočíselnou hodnotu odmocniny. Tato metoda byla zvolena z důvodu, že se v algoritmu využívají pouze celočiselné formáty hodnot a isqrt je mnohem rychlejší, než standartní funkce sqrt. Po stanovení

amplitudy na indexu k se výsledek uloží do paměti a dodatečně odečte spektrální hodnota audio kanálu (obecně řečeno odečtě se šum vzniklý nepřesností AD převodníku a mikrofonu získaného nahráním v tichém prostředí).

Blok 4.4 obsahuje kód výpočtu FFT. Výpočet je rozdělen na tři fáze. Při bližším pohledu na algoritmus jsou rozdíly v těchto fázích zřejmé. První fáze výpočtu obsahuje nejmenší množství násobení a sčítání, proto je výhodné ji oddělit od třetí fáze výpočtu, kde se nachází mnohem více matematických operací. Druhá fáze výpočtu má stejný důvod, akorát výpočtů je zde více, než ve fázi jedna, ale stále méně, než ve fázi tři. Proměnné c a s jsou hodnoty sin a cos předvypočtené v lookup tabulce ve 12 signed bitové podobě (rozsah je tedy -4096 pro -1 a 4096 pro 1). Pro zachování matematické koreknosti je tedy nutné po násobení s touto hodnotou provést binární posun do prava o 12 pozic.

Například hodnota 964*0,095=91,58 a v podobě fixní čárky (964*389)>>12=91. Na předchozím příkladu je patrná vzniklá chyba použitím fixní čárky. Pro zmenšení této chyby bychom museli zvýšit rozlišení fixní čárky. Zvýšením rozlišení riskujeme přetečení proměnné pro uložení výsledku. Tuto proměnnou bychom mohli změnit datový typ na větší, ale tím bychom zvýšili spotřebu paměti RAM a výpočetní náročnost.

Pomocí simulátoru můžeme přibližně[11] stanovit dobu potřebnou pro výpočet FFT, popřípadě pozorovat změny při úpravě programu. V případě rozdělení výpočtu na tři fáze je výsledný počet potřebných cyklů[12] 45258, v případě dvou fází (první, druhá a třetí sjednocena) je potřeba 47224 cyklů. V případě sjednocení do jednoho celku je potřeba 49216 cyklů. Rozdíl není příliš razantní, ale pokud se tento výpočet opakuje 64x za sekundu, ušetří se až 253312 cyklů, které lze využít jiným způsobem.

4.3 Výpočet příznakového vektoru

11 Jedná se o spuštění programu na „virtuálním“ mikroprocesoru. Z tohoto důsledku nelze časy považovat za odpovídající reálné aplikaci při běhu na MCU.

12 Cyklus přesněji instrukční cyklus je označení procesoru pro zpracování jedné instrukce procesoru [2].

```
//Vypocet melovskych filtru
for(j=0; j<priznaky; j++){
  ind=melPointer[j][0];
  vysledekMel[j]=0;
  for(k=0; k<melPointer[j][1]; k++, ind++, tab++)
    vysledekMel[j]+=(vysledek[ind]*melBank[tab]);
  vysledekMel[j]>>=14;
  if(maximum<vysledekMel[j])
    maximum=vysledekMel[j];
}
//▼▼▼ Normalizace na 1 bajt ▼▼▼
if(maximum>250){
  maximum=(maximum/250)+1;
  vysledekMel[priznaky]=(maximum&0x0000FF00)>>8;
  vysledekMel[priznaky+1]=maximum&0x000000FF;
}else{
  vysledekMel[priznaky]=0;
  vysledekMel[priznaky+1]=1;
}
for(j=0; j<priznaky; j++)
  vysledekMel[j]=vysledekMel[j]/maximum;
//▲▲▲ ********************* ▲▲▲
//Vypocet DCT z vysledku nenormalizovaneho mel vektoru
for(j=0; j<12; j++){
  vysledekDCT[j]=0;
  for(k=0; k<12; k++){
    vysledekDCT[j]+=vysledekMel[k]*DCT_konst[j][k];
  }
  if(maximum<abs(vysledekDCT[j]))
    maximum=abs(vysledekDCT[j]);
}
//▼▼▼ Normalizace na 1 bajt s posunutím nuly na 125 ▼▼▼
if(maximum>250)
  maximum=(maximum/125)+1;
else
  maximum=1;
for(j=0; j<priznaky; j++)
  vysledekDCT[j]=(vysledekDCT[j]/maximum)+125;
vysledekDCT[priznaky]=vysledekMel[priznaky];
vysledekDCT[priznaky+1]=vysledekMel[priznaky+1];
//▲▲▲ ************************************** ▲▲▲
for(j=0; j<(priznaky+2); j++)
  melHistory[dokonceniMel][j]=(uint8_t)vysledekDCT[j];
```

Blok 4.5: Algoritmus pro výpočet vektoru příznaků

Výpočet příznakového vektoru běží bezprostředně po provedení výpočtu FFT (ve stejné funkci). Největší problematikou je výpočet Melových koeficientů. Nejjednodušší způsob je přenásobení spektra v amplitudě vektory se samými nulami kromě oblasti, kde se nachází daný filtr a z výsledku udělat sumu. Tento způsob je sice implementačně jednoduchý, ale zbytečně výpočetně náročný. Hodnoty, které jsou násobené nulou, budou vždy rovny nule. Proto je výhodnější poznamenat indexy, kde je začátek filtru a délku tohoto filtru. Na základě těchto hodnot poté násobit hodnoty amplitudy spektra a sčítat je na danou pozici.

Algoritmus je v bloku 4.5. Pro výpočet se opět využívá předvypočtených hodnot v lookup

tabulce. Pro výpočet Melových filtrů je využito rozlišení o velikosti 14 bitů a pro výpočet DCT[13] 12 bitů. Výsledek je uložen do 2D pole o velikosti 128x14 s datovým typem uint8 (byte). Aby nedošlo k přetečení datového typu (Byte 8-bit) je výsledek vydělen maximální hodnotou nacházející se v příznakovém vektoru, pak vydělen 250 v součtu s 1 tedy hodnotou X_{max} viz (4.1). Hodnotou X_{max} je tedy "normalizován" příznakový vektor. Největší hodnota ve výsledném vektoru má tedy hodnotu 249. Poslední dva bajty příznakového vektoru jsou vyhrazeny pro dodatečné proměnné (aktuálně jsou využity pro uložení hodnoty, kterou se normalizovala energie). Celkový výsledek po provedení DCT je uložen do proměnné melHistory. Tato proměnná je 2D pole obsahující až 128 příznakových vektorů, to znamená 2 sekundy.

```
const uint8_t melPointer[12][2]={
      {
            1,7
      },{
            4,9
      },{
            8,10
      },{
            13,12
      },{
            18,14
      },{
            25,15
      },{
            32,18
      },{
            40,21
      },{
            50,24
      },{
            61,29
      },{
            74,33
      },{
            90,37
      }
};
```

```
const uint16_t melBank[230]={
4673,9346,14019,14390,10353,6317,2281,
1993,6030,10066,14102,14867,11381,7895,4408,922,
1516,5002,8488,11975,15461,14169,11158,8146,5135,2124,
2214,5225,8237,11248,14259,15617,13016,10415,7814,5213,
2612,11,
766,3367,5968,8569,11170,13771,16372,14147,11900,9654,7
407,5160,2914,667,
2236,4483,6729,8976,11223,13469,15716,15020,13079,11139
,9198,7258,5317,3377,1436,
1363,3304,5244,7185,9125,11066,13006,14947,15948,14272,
12596,10920,9244,7568,5892,4216,2540,864,
435,2111,3787,5463,7139,8815,10491,12167,13843,15519,15
682,14234,12787,11339,9891,8444,6996,5548,4100,2653,120
5,
701,2149,3596,5044,6492,7939,9387,10835,12283,13730,151
78,16174,14924,13673,12423,11173,9922,8672,7421,6171,49
20,3670,2419,1169,
209,1459,2710,3960,5210,6461,7711,8962,10212,11463,1271
3,13964,15214,16314,15233,14153,13073,11993,10913,9833,
8753,7673,6593,5513,4433,3353,2273,1193,113,
69,1150,2230,3310,4390,5470,6550,7630,8710,9790,10870,1
1950,13030,14110,15190,16270,15548,14615,13682,12750,11
817,10884,9951,9018,8085,7152,6219,5286,4353,3421,2488,
1555,622,
835,1768,2701,3633,4566,5499,6432,7365,8298,9231,10164,
11097,12030,12962,13895,14828,15761,16115,15309,14504,1
3698,12892,12086,11281,10475,9669,8863,8057,7252,6446,5
640,4834,4028,3223,2417,1611,805,229,
};
```

Blok 4.6: Náhled do lookup tabulky pro mel filtry

$$X_{max} = \frac{max(vysledek)}{250} + 1 \quad (4.1)$$

V případě výpočtu DCT se přeskočila binární operace posunu pro správný matematický výsledek.

13 DCT – discrete cosine transform. Užívané pro výpočet kepstra

Toto bylo možné z důvodu, že je poté aplikovaná normalizace. Pokud by se využilo binárního posunu mohli bychom snížit rozlišení výsledku.

Blok 4.6 obsahuje kousek lookup tabulky obsahující hodnoty pro výpočet Melových koeficientů. Proměnná melPointer obsahuje hodnoty, jako index na pozici začátku filtru do proměnné melBank (pozice [x][1]) a délku filtru (pozice [x][2]). Proměnná melBank je poté jeden velký vektor obsahující hodnoty filtrů seřazené za sebou bez nul. Podobným způsobem je uložena i tabulka pro DCT, ale bez nutnosti indexů pro odkazování.

4.4 Výpočet DTW

Výpočet DTW je algoritmicky velmi jednoduchý. V základě se jedná o nalezení cesty s minimální cenou (popřídě maximální). Nejjednodušší implementace bez analýzy cesty, ale pouze výsledné ceny je v bloku 4.7. Jedná se o výpočet probíhající přes odkazy do pole o velikosti m x n. Funkce d() je funkcí vzdálenosti. Tato vzdálenost může mít podobu Euklidovské vzdálenosti (4.2), vzdálenosti Manhattan (4.3), Chebyshev (4.4), atd.

$$d(p,q)=\sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \quad (4.1)$$

$$d(p,q)=\sum_{i=1}^{n}|p_i-q_i| \quad (4.2)$$

$$d(p,q)=max_i(|p_i-q_i|) \quad (4.3)$$

Nejčastěji volená funkce vzdálenosti je Euklidovská popřípadě city block (Manhattan). Výpočet běží iterativně přes všechny indexy v poli. V každém kroku se nalezne minimální hodnota z předchozích kroků, ke které se přičtě vzdálenost aktuálně zvolených vektorů.

```
int DTWDistance(s: array [1..n], t: array [1..m]) {
    DTW := array [0..n, 0..m]
    for i := 1 to n
        DTW[i, 0] := infinity
    for i := 1 to m
        DTW[0, i] := infinity
    DTW[0, 0] := 0
    for i := 1 to n
        for j := 1 to m
            cost := d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],    // insertion
                                        DTW[i  , j-1],    // deletion
                                        DTW[i-1, j-1])    // match
    return DTW[n, m]
}
```

Blok 4.7: Jednoduchá implementace DTW [4.3]

Tato jednoduchá implementace je ovšem velmi náročná na paměť. Pro každý vzor potřebujeme samostatné pole pro ukládání mezivýsledků. V případě, že nepotřebujeme analyzovat cestu kterou byla dosažena minimální cena, máme příliš mnoho zbytečných dat v historii. Problém se zbytečnými daty lze jednoduše vyřešit, implementací pouze 1D pole o velikosti odpovídající délce vzorových dat, do kterého se postupným přepisováním hledá nejlepší cesta.

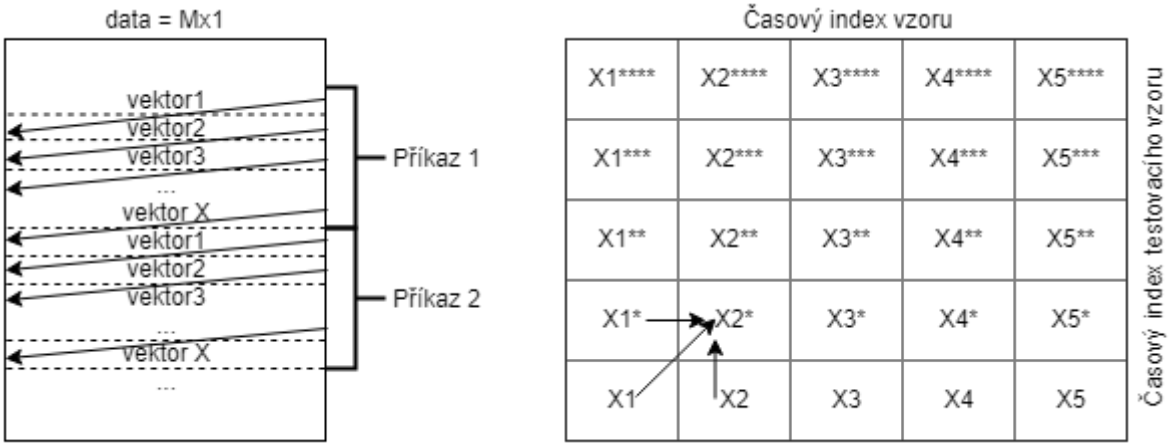
```
void DTW(uint8_t* data, uint32_t poziceMel, uint32_t indexDTW, uint32_t delkaDTW,
uint32_t zpracovanoDTW, uint32_t pos){
  uint32_t konst=(delkaDTW*2)/4;
  uint32_t tempA=zpracovanoDTW;
  uint32_t ddd=small2(delkaDTW,konst+tempA);
  uint32_t bbb=1;
  if((zpracovanoDTW)>konst)
    bbb=(tempA)-konst;
  if(zpracovanoDTW==0){
    DTWVect[indexDTW]=diff8bit(melHistory[poziceMel],data,pos);
    for(uint32_t i=1, x=data+priznakyE, p=indexDTW; i<ddd; i++, x+=priznakyE, p++){
      DTWVect[p+1]=DTWVect[p]+diff8bit(melHistory[poziceMel],x,pos);
    }
  }else{
    uint32_t temp=(poziceMel+zpracovanoDTW)%128;
    dtw0=DTWVect[indexDTW];
    temp0=dtw0+diff8bit(melHistory[temp],data,pos);
    DTWVect[indexDTW]=temp0;
    for(uint32_t i=1, x=data+priznakyE, p=indexDTW+1; i<ddd; i++, p++, x+=priznakyE){
      dtw1=DTWVect[p];
      if(i<bbb)
        DTWVect[p]=tE;
      else
        DTWVect[p]=small3(temp0,dtw0,dtw1)+diff8bit(melHistory[temp],x,pos);
      dtw0=dtw1;
      temp0=DTWVect[p];
} } } }
```

Blok 4.8: Implementace DTW s úsporou paměti

Vzhledem k neznámé velikosti vzorů je problém vytvořit datovou strukturu vhodnou k ukládání mezivýsledků. Nejjednodušší by bylo využít dynamického pole, ale tento přístup je v našem případě problematický. Nelze totiž odhadnout kolik místa v paměti by se využilo. Z tohoto důvodu je deklarované pole o pevně dané velikosti pro ukládání mezivýsledků DTW. S tímto polem spolupracuje 2D pole obsahující indexy pro stanovení pozic jednotlivých vzorů.

Implementace v předchozím bloku (blok 4.8) je vytvořena tak, aby ji bylo možné využít v reálném čase. Hodnoty, které byly vypočteny v posledních 128 mikrosegmentech jsou uloženy v poli melHistory. Zavoláním funkce DTW tedy můžeme vypočítat podobnost s posledně získaným mikrosegmentem nebo porovnat s mikrosegmentem, který byl získán před 126 mikrosegmenty. Proměnná data je ukazatelem do lookup tabulky, ve které jsou uložena vzorová data. Vzorová data se nachází v 1D poli, proto je správný index kriticky důležitý pro správnou funkci Výpočet DTW je rozdělen na dvě fáze. Při první fázi dojde k počátečnímu výpočtu, kdy není zapotřebí nalézt minimální hodnoty z předchozích kroků, jelikož začínají všechny na hodnotě TOP. Druhá fáze je iterace přes všechny vektory v testovacím vzoru, kdy se vypočte vzdálenost a přičte se k ní minimální hodnota předchozího kroku. Funkce diff8bit() je funkce zajišťující výpočet vzdálenosti dvou vektoru pomocí vztahu (4.2). Funkce small3() je funkce vracející nejmenší hodnotu ze tří prvků.

Algoritmus tedy běží postupně a při každém zavolání by měla být navýšena proměnná zpracovanoDTW, která zajišťuje ukazatel pro vyhodnocovaná data. Proměnná indexDTW a delkaDTW je využita jako index a délka do pole DTWVect, což je proměnná ukládající mezivýsledy pro vyhodnocení DTW. V algoritmu je zajištěna i mez, ve které by se měly pohybovat časové indexy pro trénovací a testovací data zajištující omezení rozptylu v čase (v ideálním případě by cesta v tabulce DTW měla vést na úhlopříčce).



Obrázek 4.2: Znázornění seřazení vzorových dat (vlevo) a tabulka pro DTW (vpravo)

Na obrázku 4.2 je znázorněna datová struktura lookup tabulky s trénovacími vzory. Tato tabulka má dimenzi Mx1 (tedy 1D pole). Kde M vyjadřuje součet délek jednotlivých vzorů vynásobené velikostí příznakového vektoru viz. (4.4). V tomto vztahu má X význam počtu trénovacích vzorů, K počet příznakových vektorů v daném vzoru a R je velikost příznakového vektoru.

$$M = \sum_{m=0}^{X} \sum_{i=0}^{K} v_m(i) * R \quad (4.4)$$

V obrázku na pravé straně je jednoduše znázorněna tabulka pro výpočet DTW. Hodnota X_y je hodnota na počátku algoritmu (v čase 0). Dále Hodnoty X_y*, X_y**, … jsou výsledky v časech 1,2,... Šipky jsou názorněním hodnot, ze kterých se vybere minimum pro vyhodnocení ukazovaného prvku v tabulce. Vztah (4.5) je výpočet následující prvku.

$$\begin{aligned} X_I^{\mathrm{T}} &= d\left(v_T(I), v_R(I)\right) + min\left(X_{(I-1)}^{\mathrm{T}}, X_{(I-1)}, X_I\right) \\ specificky\ X_2^* &= d\left(v_T(2), v_R(2)\right) + min\left(X_1^*, X_{1,} X_2\right) \end{aligned} \quad (4.5)$$

Literatura

[4.1] *Atmel Studio 7.* www.microchip.com [online]. [cit. 14.3.2018] Dostupné na WWW: https://www.microchip.com/avr-support/atmel-studio-7

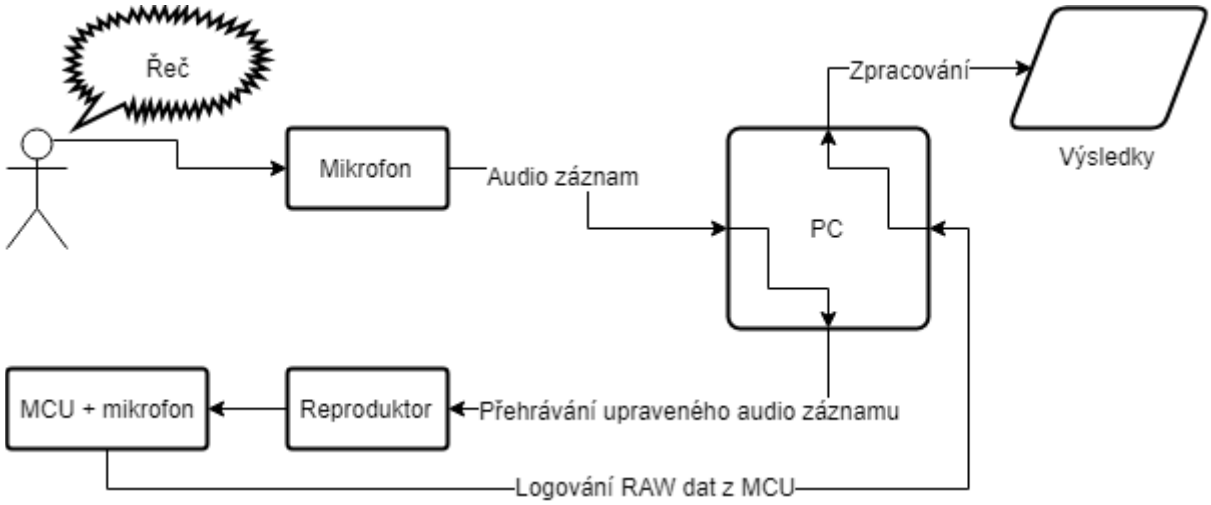
[4.2] *Instrukce a instrukční cyklus.* is.mendelu.cz [online]. [cit. 12.3.2018] Dostupné na WWW: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=646

[4.3] Dynamic time warping. en.wikipedia.org [online]. [cit. 14.3.2018] Dostupné na WWW: https://en.wikipedia.org/wiki/Dynamic_time_warping

5. Výsledky

Ověření hodnot dosažených výsledků přesnosti je náročné z důvodu, že aplikace běží na samostatné platformě. Nelze kvalitativně otestovat změnu v přesnosti při změně jistých parametrů v algoritmu. Důvodem je, že při opakované, ačkoliv pečlivé mluvě, nikdy neřekneme jeden příkaz stejně. Pokud tedy změníme parametry a vyzkoušíme jestli se změnila přesnost, nevíme jestli je to z důvodu lepší promluvy nebo lepšího parametru.

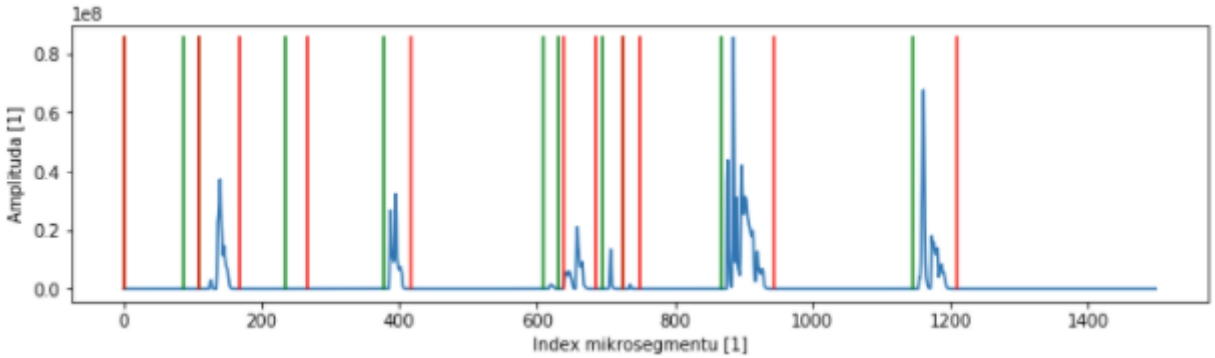
Nicméně v algoritmu MCU je rutina, která zajišťuje odesílání dat v reálném čase pomocí sériové linky. Do těchto dat se řadí průběh signálu tak, jak byl zaznamenaný. Dále výsledek FFT pro poslední mikrosegment a MFC koeficienty. Můžeme tedy tuto rutinu spustit a zaznamenávat data do souboru na počítači. Zaznamenaný signál je možné poté zpracovat externím scriptem (programem), který bude provádět identické operace jako MCU co se výpočtů týče (operace s fixní čárkou, zaokrouhlování, lookup tabulky, …). Při změně parametru například v proceduře DTW nedojde k žádné změně testovaného signálu, ale algoritmu samotného a my poté pouze sledujeme změny v přesnosti.



Obrázek 5.1: Blokové schéma testovacího "obvodu"

Pomocný script je přepsaný program z jazyka C do jazyka Python. Procedura je velmi jednoduchá. Načte se soubor se zaznamenaným zvukem v časové oblasti, poté se aplikuje detekce prahu energie a vypočte FFT. V momentě, kdy dojde k překročení prahu energie, zahájí se výpočet příznakových vektorů a spustí se rozpoznávání příkazu stejnou metodou DTW jako v MCU a stejně se i ukončí. Script pouze sbírá data o časech, kdy došlo k detekci slova a jakého slova. Tyto údaje se uloží a vzhledem k tomu, že je známo kdy se jaký příkaz vyslovil, můžeme poté na základě těchto dat provést vyhodnocení.

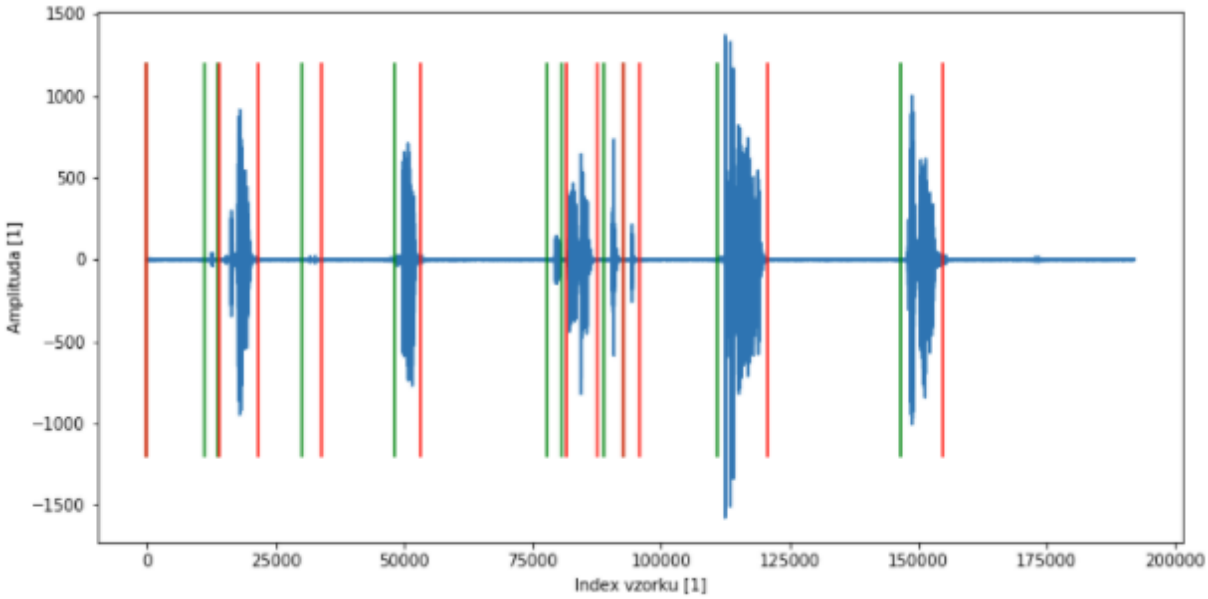
Aby byly zajišteny přesné časové indexy příkazů, nahrál se hodinový audio záznam na externím zařízení s několika příkazy na předem daných časových indexech. Tento audio záznam se pomocí reproduktorů přehrál nahlas a během tohoto přehrávání se "stahovali" data z MCU. Díky tomuto přístupu lze záznam dat z MCU zopakovat, například v tichém/zašumnělém prostředí a promluva bude stále stejná. Nastává zde však problém se zkreslením audio signálu. Nahraný záznam je postižen chybou mikrofonu (dynamikou, zesílením, …), chybou vzniklou přehráváním pomocí reproduktoru (opět rozdílná dynamika) a výsledné zpracování pomocí mikrofonu na platformě MCU. Tento fakt je nutné brát na zřetel.



Obrázek 5.2: Detekční prahy a zobrazení energie

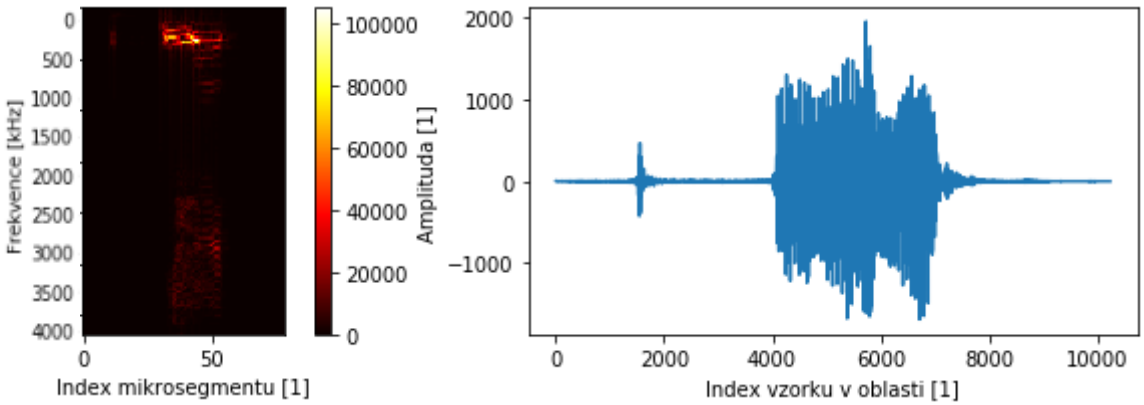
Jedna z výhod zpracování pomocí Python skriptu je možnost jednoduché vizualizace funkce algoritmů. Obrázek 5.2 znázorňuje detekci řeči na základě změny energie signálu, kde zelená čára je začátek promluvy a červená čára je konec promluvy. Obrázek 5.3 zobrazuje stejné prahy, ale s vyobrazením reálného signálu. Obrázek 5.4 znázorňuje spektrum (vlevo) úseku signálu a signál samotný (vpravo).

Je viditelné, že mezi indexy 600 a 800 je detekováno více řečových úseků, než by mělo být. Ideálně by se mělo jednat o jeden celý úsek. Toto je vzorový příklad problému nastavení parametrů, které chceme zlepšit.



Obrázek 5.3: Detekční prahy a zobrazení signálu

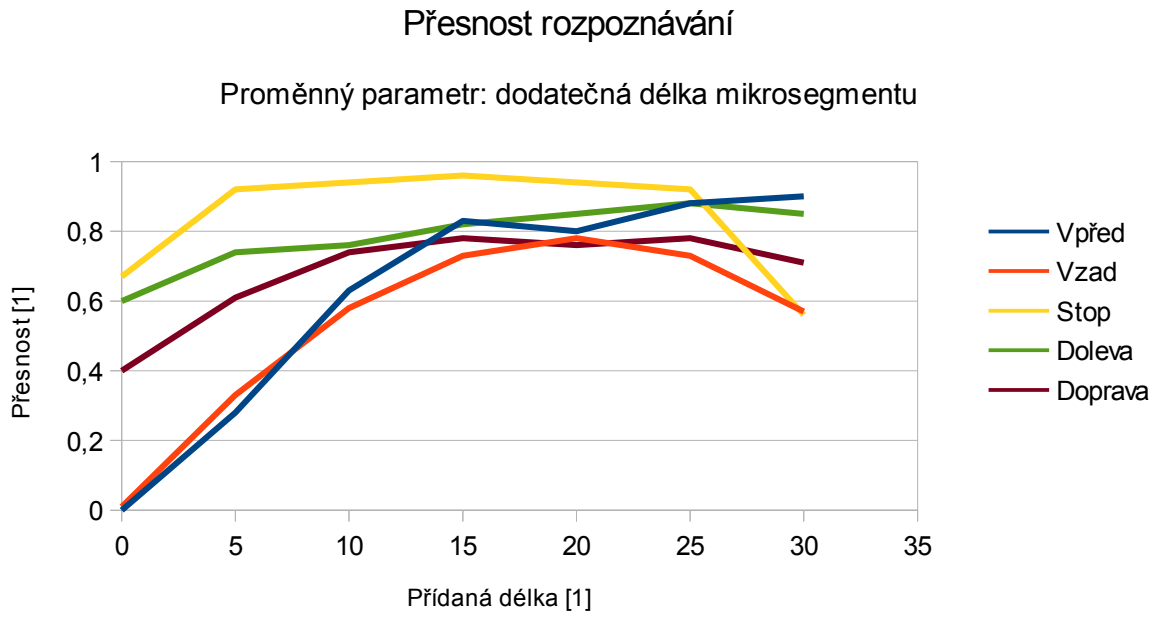
Předchozí obrázek znázorňuje aktivaci rozpoznávání řeči i při velmi malých změnách signálu (první oblast zleva okolo pozice 12500, pak třetí oblast okolo pozice 35000).



Obrázek 5.4: Spektrum zobrazené heatmapou (vlevo) a odpovídající úsek signálu (vpravo)

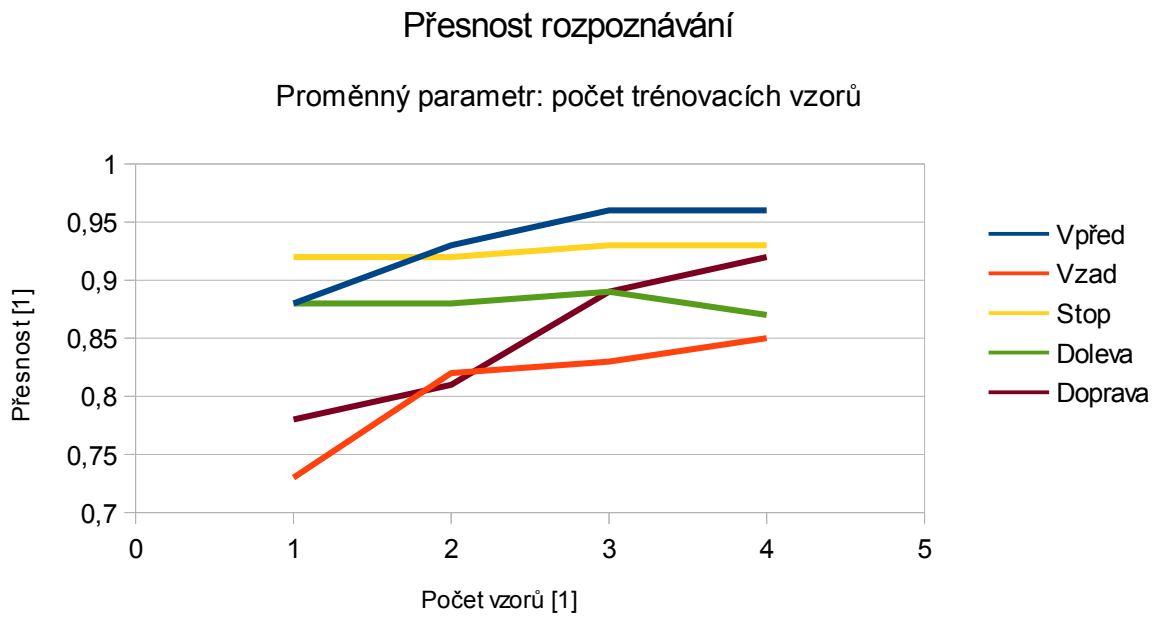
Spektrum je zobrazené pomocí spectrogramu v barvách "heatmapy". Na horizontální ose je index mikrosegmentu a na vertikální ose je frekvence.

První testovaný parametr který lze nastavit, je rozšíření mikrosegmentu o X okének na začátku i na konci promluvy. V momentě, kdy dojde k překročení prahu energie pro detekci promluvy, se začne analyzovat mikrosegment o -X indexů. Tento parametr je velmi důležitý, neboť při detekci energie můžeme přijít o data na začátku i na konci promluvy. Jinými slovy, dle obrázku 5.3 dojde k posunutí zeleného prahu a červeného prahu podle nastaveného parametru (vyšší hodnota zelený práh se posune doleva a červený práh doprava a naopak). Přesnost rozpoznávání je testovaná pouze s pěti příkazy a to: vpřed, vzad, stop, doleva, doprava.



Obrázek 5.5: Nastavení parametru dodatečná délka mikrosegmentu

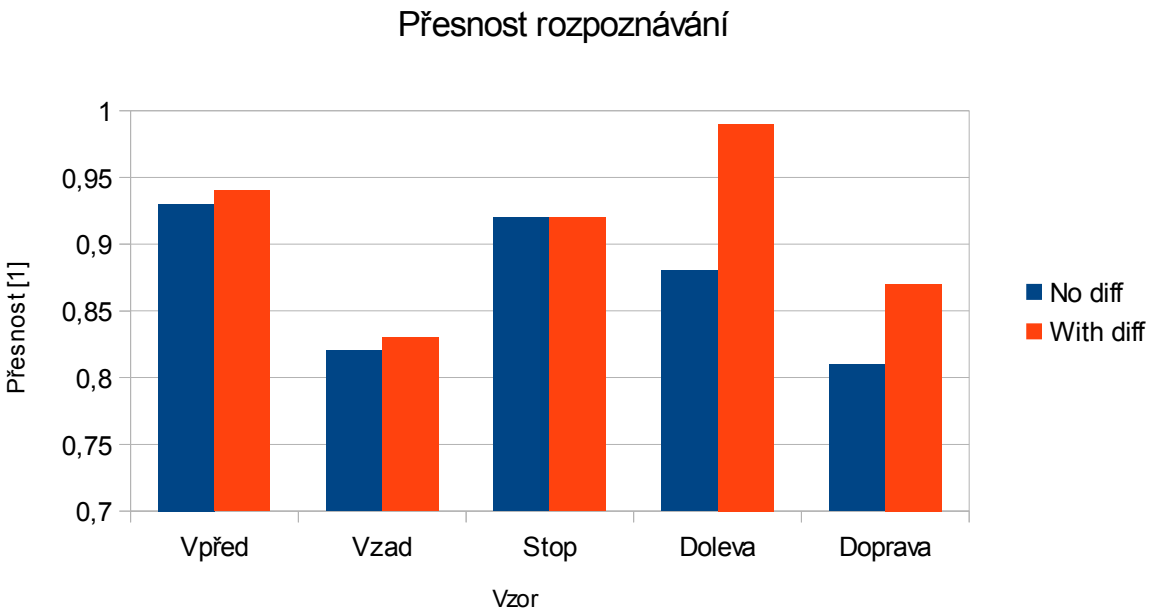
Předchozí obrázek je znázornění změny v přesnosti rozpoznávání při změně parametru. Je viditelné, že při malé hodnotě je přesnost malá a při příliš vysoké hodnotě přesnost také postupně klesá. Nehledě na fakt, že dojde ke zmenšení rozsahu dat k analýze.



Obrázek 5.6: Změna přesnosti při změně počtu vzorů

Obecně platí, čím více trénovacích dat k analýze (vzorů), tím lze dosáhnout větší přesnosti rozpoznávání. Bohužel MCU má omezenou paměť a lze na něj nahrát omezené množství trénovacích vzorů. Obrázek 5.6 ilustruje zvyšující se přesnost rozpoznávání pro 1 až 4 trénovací vzory. Ostatní parametry jsou konstantní. Trend by měl mít tendenci růstu, ale vzniká zde otázka

kvality nahraného trénovacího vzoru. Pokud je horší vzor, bude horší výsledek.



Obrázek 5.7: Změna přesnosti změnou příznakového vektoru o differenci mezi mikrosegmenty

Na předchozím obrázku je znázornění rozdílu mezi příznakovými vektory s differencí mezi sousedními mikrosegmenty a bez difference. Největší rozdíl je v příkazech doleva a doprava. Vysledné parametry pro maximalizaci přesnosti jsou následující.

- Energetická hladina pro aktivaci rozpoznávání: 1,4 násobek součtu energie deseti předchozích mikrosegmentů
- Přidaný rozsah pro analyzované okénko 25
- Počet trénovacích vzorů: Z důvodu omezení MCU je počet vzorů stanoven na 2.
- Příznakový vektor s differencí.

Příkaz	Přidaná délka							Počet vzorů				Difference	
	0	5	10	15	20	25	30	1	2	3	4	Ne	Ano
Vpřed	0%	28%	63%	83%	80%	88%	90%	88%	93%	96%	96%	93%	94%
Vzad	1%	33%	58%	73%	78%	73%	57%	73%	82%	83%	85%	82%	83%
Stop	67%	92%	94%	96%	94%	92%	56%	92%	92%	93%	93%	92%	92%
Doleva	60%	74%	76%	82%	85%	88%	85%	88%	88%	89%	87%	88%	99%
Doprava	40%	61%	74%	78%	76%	78%	71%	78%	81%	89%	92%	81%	87%

Tabulka 5.1: Souhrnné výsledky testování

Pomocí optimalizace parametrů algoritmu jsme dosáhli souhrnné přesnosti zobrazené v tabulce 5.1. Červené sloupce jsou hodnoty odpovídající zvolenému parametru. Počet testovaných příkazů je pět, ale je samozřejmě možné jich nahrát více (limit je nastaven na 12 vzorů s průměrnou délkou 53 mikrosegmentů). Nejdříve se optimalizovala přidaná délka, poté počet vzorů a nakonec se otestovala přesnost s differencí mezi mikrosegmenty.

Závěr

Rozpoznávání hlasu je náročné na výpočetní výkon a paměť zařízení na němž systém běží. Nicméně pro jednoduché příkazy je možné vyvinout systém pro rozpoznávání jednoduchých příkazů jako jsou čísla a jednoslovné příkazy.

Systém navržený a otestováný v této práci dosahuje relativně vysoké kvality, kterou by bylo možné ještě znatelně zvýšit. Například instalací kvalitnějšího mikrofonu s nízkým zatížením šumu, zvýšením přesnosti v algoritmech atd. Výsledný rozpoznávací systém hlasu je závislý na řečníkovy a byl by vhodný pro nasazení například do hraček, pomocných systému nebo jako sekundární zabezpečení v zabezpečovacích systémech.

Seznam veškeré literatury a zdrojů

[1.1] Yundong Zhang, Naveen Suda, Liangzhen Lai, Vikas Chandra , *Hello Edge: keyword spotting on microcontrollers* [online]. [cit. 22.3.2018]. Dostupné na WWW: https://arxiv.org/abs/1711.07128

[1.2] PSUTKA, Josef. MÜLLER, Luděk. MATOUŠEK, Jindřích. RADOVÁ, Vlasta. *Mluvíme s počítačem česky*. 1 vydání. Praha: nakladatelství Academia, 2006, 738 stran. ISBN 80-200-1309-1.

[2.1] *www.wikiskripta.eu* [online]. [cit. 27.1.2018]. Dostupné na WWW: https://upload.wikimedia.org/wikipedia/commons/7/7c/HumanEar.jpg

[2.2] Člověk a zvíře: Sluchový rozsah | Eduportál Techmania. *Eduportál | Eduportál Techmania* [online]. Copyright © Techmania Science Center, o.p.s. [cit. 05.04.2018]. Dostupné na WWW: http://edu.techmania.cz/cs/katalog/clovek-zvire/316/sluchovy-rozsah

[2.3] *wiki.metropolia.fi* [online]. [cit. 27.1.2018]. Dostupné na WWW: http://hades.mech.northwestern.edu/images/thumb/8/80/ElectretMicrophone.gif/300px-ElectretMicrophone.gif

[2.4] ŠERÝCH, Jakub. *cs.wikipedia.org* [online]. [cit. 27.2.2018]. Dostupné na WWW: https://upload.wikimedia.org/wikipedia/commons/e/ed/Kvantov%C3%A1n%C3%AD.png

[2.4] Field-effect transistor - Wikipedia. [online]. [cit. 27.1.2018] Dostupné na WWW: https://en.wikipedia.org/wiki/Field-effect_transistor

[2.5] PSUTKA, Josef. MÜLLER, Luděk. MATOUŠEK, Jindřích. RADOVÁ, Vlasta. 4.3 Zpracování v časové oblasti. *Mluvíme s počítačem česky*. 1 vydání. Praha: nakladatelství Academia, 2006, 738 stran. ISBN 80-200-1309-1.

[2.5] Asymptotická složitost. w*ww.algoritmy.net* [online]. [cit. 06.03.2018]. Dostupné na WWW: https://www.algoritmy.net/article/102/Asymptoticka-slozitost

[2.6] Cooley-Turkey FFT algorithm. *en.wikipedia.com* [online]. [cit. 6.3.2018]. Dostupné na WWW: https://upload.wikimedia.org/wikipedia/commons/c/cb/DIT-FFT-butterfly.png

[2.7] VEDALA, Krishna. *cs.wikipedia.org* [online]. [cit. 27.2.2018]. Dostupné na WWW: https://commons.wikimedia.org/wiki/File:Mel-Hz_plot.svg

[2.8] Practical Cryptography. *Practical Cryptography* [online]. Copyright © 2009 [cit. 03.04.2018]. Dostupné na WWW: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

[2.9] PSUTKA, Josef. MÜLLER, Luděk. MATOUŠEK, Jindřích. RADOVÁ, Vlasta. 4.6.4 Melovské kepstrální koeficienty. *Mluvíme s počítačem česky*. 1 vydání. Praha: nakladatelství Academia, 2006, 738 stran. ISBN 80-200-1309-1.

[2.10] DTW warping path. Dynamic Time Warping Techniques for Missing Value. [online] cst.tu-plovdiv.bg [cit. 27.2.2018]. dostupné na WWW: http://cst.tu-plovdiv.bg/bi/DTWimpute/DTWalgorithm.html

[3.1] Wideband audio - Wikipedia. [online]. [cit. 22.03.2018] Dostupné na WWW: https://en.wikipedia.org/wiki/Wideband_audio

[3.2] General information - - UC3-L0 Xplained. *Home* | *Microchip Technology Inc.*[online]. [cit. 20.02.2018] Dostupné na: https://www.microchip.com/webdoc/uc3l0xplained/uc3l0xplained.chapter.poj_mfp_kc.html

[3.3] Response first-order low-pass filter. Wikipedia. [online].[cit. 22.03.2018] Dostupné na: https://en.wikipedia.org/wiki/Low-pass_filter

[3.4] Passive Low Pass Filter.. ElectronicsTutorials. [online]. [cit. 22.03.2018]. Dostupné na: https://www.electronics-tutorials.ws/filter/filter_2.html

[3.5] MCE100 NEW2 | GM electronic, spol. s.r.o.. *GM electronic* | *elektronické součástky, komponenty* . | *GM electronic, spol. s.r.o.* [online]. [cit. 20.02.2018] Dostupné na: https://www.gme.cz/mce100-new2

[3.6] Atmel Corporation 2012. Tabulka 2-1 str. 6. [online]. [cit. 22.03.2018] Dostupné na: http://ww1.microchip.com/downloads/en/DeviceDoc/doc32099.pdf

[4.1] *Atmel Studio 7.* www.microchip.com [online]. [cit. 14.3.2018] Dostupné na WWW: https://www.microchip.com/avr-support/atmel-studio-7

[4.2] *Instrukce a instrukční cyklus.* is.mendelu.cz [online]. [cit. 12.3.2018] Dostupné na WWW: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=646

[4.3] Dynamic time warping. en.wikipedia.org [online]. [cit. 14.3.2018] Dostupné na WWW: https://en.wikipedia.org/wiki/Dynamic_time_warping

Seznam obrázků, diagramů, kódů a tabulek

Seznam příloh

Veškeré přílohy jsou na přiloženém CD včetně seznamu obsahu: soubor readme.txt.