VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

ONLINE DETEKCE JEDNODUCHÝCH PŘÍKAZŮ V AUDIOSIGNÁLU

ONLINE DETECTION OF SIMPLE VOICE COMMANDS IN AUDIOSIGNAL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. MIROSLAV ZEZULA

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. JIŘÍ KREJSA, Ph.D.

BRNO 2011

Zadání

Charakteristika problematiky úkolu:

Navrhněte a implementujte metodu online detekce hlasových příkazů v audiosignálu.

Cíle diplomové práce:

1. zpracujte extrakci rysů signálu pomocí spektrální analýzy
2. zpracujte metodu pro rozpoznávání vzorů
3. implementujte metodu rozpoznávání zadaných vzorů s co nejmenší výpočetní náročností

Abstrakt

Tato práce popisuje vývoj hlasového modulu, který je schopen rozpoznávat jednoduché řečové povely na základě porovnání zvukového vstupu s uloženými vzory. První část práce obsahuje popis použitého algoritmu a ověření jeho funkčnosti. Algoritmus je založen na Mel-frekvenčních cepstrálních koeficientech a dynamickém borcení času. Dále je navržen hardware hlasového modulu, obsahující signálový kontrolér 56F805 firmy Freescale. Signál z mikrofonu je upraven operačními zesilovači a digitálním filtrem. Třetí část se zabývá vývojem software pro kontrolér a popisuje implementaci algoritmu v pevné řádové čárce s ohledem na omezené možnosti kontroléru. Závěrečná zkouška prokazuje použitelnost modulu v prostředí s nízkým obsahem šumu.

Summary

This thesis describes the development of voice module, that can recognize simple speech commands by comparation of input sound with recorded templates. The first part of thesis contains a description of used algorithm and a verification of its functionality. The algorithm is based on Mel-frequency cepstral coefficients and dynamic time warping. Thereafter the hardware of voice module is designed, containing signal controller 56F805 from Freescale. The signal from microphone is conditioned by operational amplifiers and digital filter. The third part deals with the development of software for the controller and describes the fixed point implementation of the algorithm, respecting limited capabilities of the controller. Final test proves the usability of voice module in low-noise environment.

Klíčová slova

aliasing, dynamické borcení času, Mel-frekvenční cepstrální koeficienty, rozpoznávání řeči, signálový kontrolér 56F805, výpočty v pevné řádové čárce

Keywords

aliasing, dynamic time warping, fixed point arithmetics, Mel-frequency cepstral coefficients, signal controller 56F805, speech recognition

ZEZULA, M. *Online detekce jednoduchých příkazů v audiosignálu.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2011. 58 s. Vedoucí Ing. Jiří Krejsa, Ph.D.

Čestně prohlašuji, že jsem diplomovou práci na téma *Online detekce jednoduchých příkazů v audiosignálu* vytvořil samostatně pod vedením svého vedoucího diplomové práce s využitím odborné literatury, kterou jsem všechnu uvedl v seznamu literatury.

Bc. Miroslav Zezula

Děkuji Ing. Jiřímu Krejsovi, Ph.D. za přátelský přístup nejen při tvorbě této diplomové práce. Dále děkuji Bc. Daliboru Šulcovi za pomoc při shánění některých součástek a doc. Ing. Pavlu Vorlovi, Ph.D. za konzultaci zapojení vstupní části. A především děkuji své rodině a přátelům za podporu po celou dobu studia.

Bc. Miroslav Zezula

Obsah

1. Úvod

1.1. Motivace

Problém efektivního zadávání příkazů a dat do počítače je stejně starý, jako výpočetní technika sama. Způsob komunikace, který je přirozený pro člověka, bohužel není ve své podstatě vhodný pro počítač. Zatímco lidé se většinou dorozumívají pomocí mluvené řeči nebo písma (přičemž význam je často závislý na kontextu a tentýž obsah je možno vyjádřit mnoha způsoby), počítač pracuje s přesně definovanými elektrickými signály. Z tohoto důvodu je nutno vstupní informace nejprve převést do formy zpracovatelné počítačem.

Do prvních počítačů zadávala údaje výhradně vyškolená obsluha prostřednictvím přepínačů nebo děrné pásky. S dalším rozvojem se nedílnou součástí počítače stala klávesnice a obrazovka, později i myš nebo obdobné ukazovací zařízení. Ovládání počítače se tak přiblížilo lidem. V současné době, kdy je standardním vybavením počítače grafický operační systém, zvládne jeho obsluhu i laik.

Tím, že úloha převodu informace do vhodné formy byla přenechána počítači, vzrostly nároky na jeho výpočetní výkon. Není přehnané tvrdit, že při běžném domácím použití počítače se většina výpočetního výkonu spotřebuje na zpracování vstupu od uživatele a vykreslení výsledků, pouze malá část slouží k řešení samotné úlohy.

V současnosti se ve specializovaných oblastech začíná uplatňovat ovládání počítače pomocí hlasu. Jak bylo zmíněno na začátku, tato forma komunikace je pro člověka nejpřirozenější. To ovšem zároveň znamená, že je poměrně vzdálená počítači a zpracování hlasu proto není jednoduché. Uspokojivě je vyřešen problém přepisu mluvené řeči do podoby textu (např. [1]) a ovládání počítače pomocí omezeného souboru hlasových příkazů. Ovládání počítače pomocí zcela přirozené mluvy vyžaduje, aby počítač chápal význam jednotlivých slov, a proto zatím nebylo plně realizováno.

Možnost hlasového ovládání je atraktivní mimo jiné i v oblasti mobilní robotiky. Mobilní robot sice můžeme na dálku ovládat např. pomocí notebooku a bezdrátové sítě, schopnost zasahovat do jeho činnosti pomocí několika hlasových povelů přesto představuje určitý přínos.

Vzhledem k tomu, že mobilní robot je nejčastěji napájen z baterií a jeho rozměry i hmotnost jsou limitovány, je dostupný výpočetní výkon omezený. Hlavní počítač typu PC často slouží jen pro nejvyšší úroveň řízení (plánování pohybu), kde postačuje doba odezvy řádově ve zlomcích sekundy. Pro obsluhu jednotlivých senzorů a akčních členů je výhodnější použít několik jednoduchých mikrokontrolérů, které zajišťují rychlejší odezvu při řešení dílčích úloh a jejich připojení k běžným senzorům je snadnější. Tento přístup vede na modulární koncepci mobilního robotu, kde vždy jeden senzor tvoří spolu s příslušným mikrokontrolérem samostatný modul. Jednotlivé moduly jsou pak s hlavním počítačem propojeny pomocí vhodné sběrnice.

Od hlasového ovládání v oblasti mobilní robotiky očekáváme okamžitou odezvu, která vyžaduje zpracování vstupního signálu v reálném čase. Vzhledem k tomu, že není žádoucí

výrazně zatěžovat hlavní počítač dalšími úlohami, jeví se jako logický krok implementace hlasového ovládání do samostatného modulu podobně, jako je tomu u ostatních senzorů. Vývoj modulu pro hlasové ovládání je obsahem této diplomové práce.

1.2. Formulace cílů, postup řešení

1. Vyberte vhodnou metodu pro rozpoznávání hlasových příkazů:
   (a) popište obecný způsob rozpoznávání izolovaných hlasových příkazů
   (b) specifikujte podrobně jednotlivé části rozpoznávače
   (c) naprogramujte dle vytvořeného popisu jednoduchý rozpoznávač v prostředí MATLAB a ověřte jeho činnost pomocí vhodné sady testovacích dat
2. Navrhněte hardwarovou část hlasového modulu. Respektujte tyto požadavky:
   (a) modul bude realizován na jedné desce plošných spojů
   (b) modul bude napájen napětím 5 V, odběr modulu má být nejvýše 0,5 A
   (c) modul bude komunikovat s nadřazeným systémem po sběrnici RS-485
   (d) napájení a sběrnice RS-485 budou připojeny přes společný konektor typu RJ-14
   (e) mikrofon nebude součástí modulu, ale modul bude vybaven konektorem pro připojení externího mikrofonu
3. Navrhněte softwarovou část hlasového modulu, která bude zajišťovat tyto funkce:
   (a) rozpoznávání hlasových příkazů
   (b) učení nových hlasových příkazů
   (c) komunikaci s nadřazeným systémem

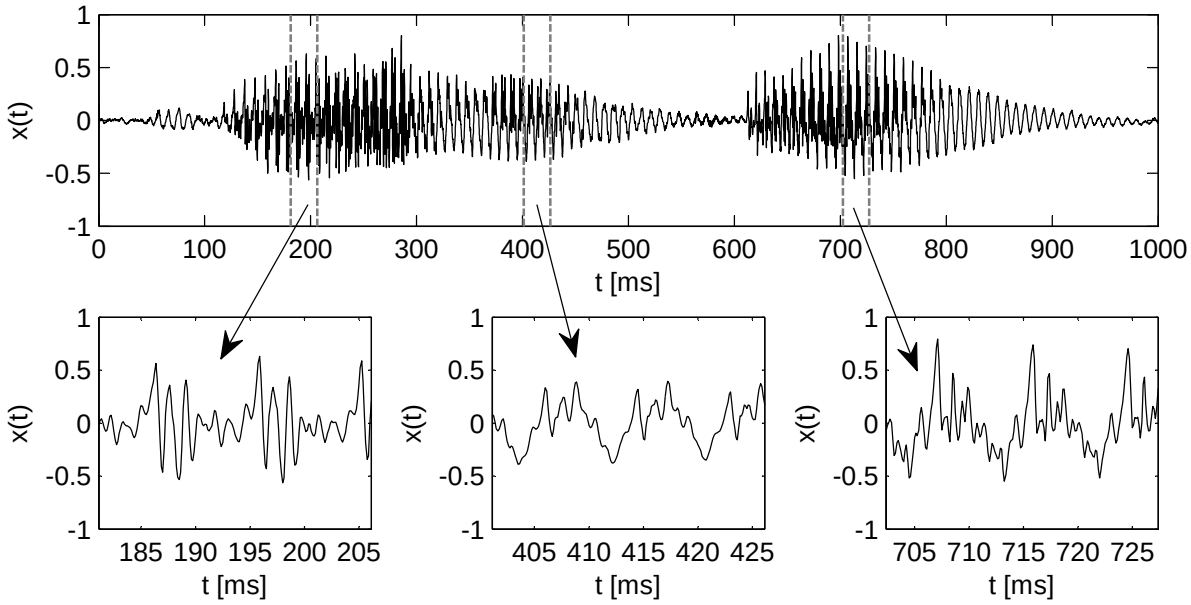
2. Popis algoritmu

V současné době jsou základní možnosti rozpoznávání hlasových příkazů dobře prozkoumány a nebylo by tedy účelné vyvíjet rozpoznávač řeči od základů. Princip, popsaný v této části, byl proto téměř doslovně převzat z [2]. Přesto považuji za nezbytné jej zde uvést, neboť je výchozím bodem pro další části práce.

2.1. Řečový signál

Pod pojmem *zvuk* rozumíme změny tlaku (oproti střední hodnotě), které se šíří pružným prostředím. Tyto změny můžeme v určitém místě snímat mikrofonem a vyhodnocovat jako signál[1], který má v čase t hodnotu $x(t)$. Lidská řeč je pak pouze specifickým případem zvuku.

Na obrázku 2.1 je příklad řečového signálu (jedná se o slovo „Vánoce“). V horní části obrázku je zobrazen celý signál, v dolní části jsou pak tři detaily. Z obrázku je patrno, že charakter signálu se z dlouhodobého hlediska (v průběhu slova) mění a jedná se tedy o nestacionární signál. V relativně krátkém časovém úseku se však signál jeví jako stacionární (či dokonce periodický).



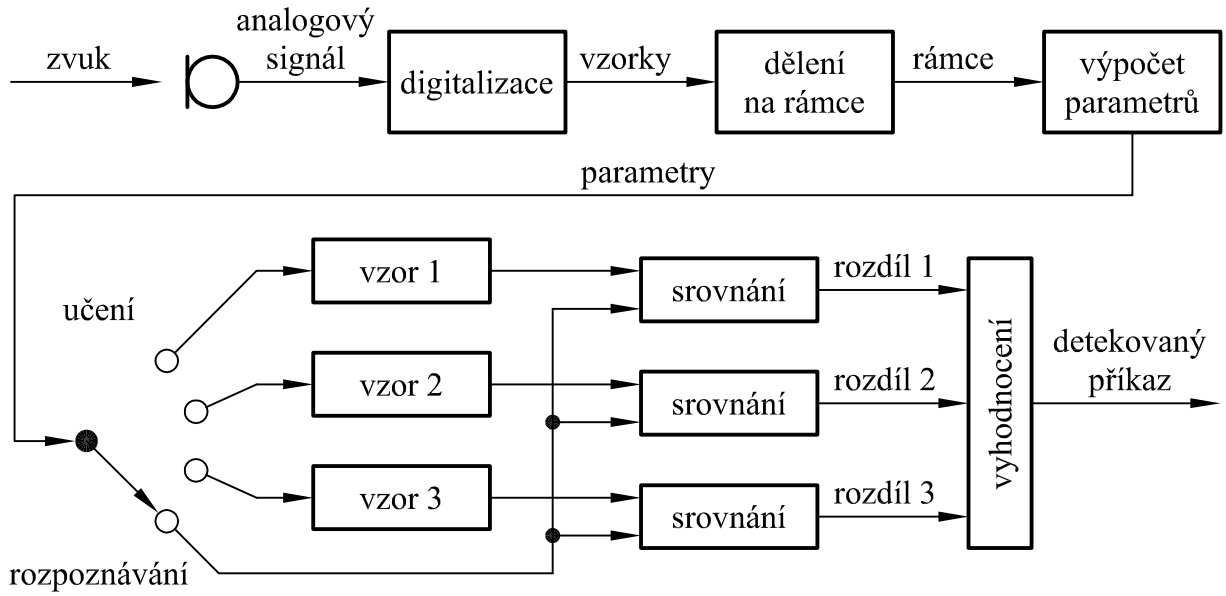
Obrázek 2.1: Příklad řečového signálu

K analýze takovéhoto druhu signálu lze použít tzv. *evoluční charakteristiky*. Hodnoty těchto charakteristik jsou funkcemi času a určíme je pro daný čas z malého úseku signálu, ve kterém lze signál považovat za stacionární. Pokud se nám podaří popsat průběh řečového signálu pomocí vhodných evolučních charakteristik, můžeme posuzovat podobnost dvou řečových signálů na základě podobnosti jejich evolučních charakteristik.

[1] Tento signál budeme z praktických důvodů považovat za bezrozměrnou veličinu, ačkoliv vyjadřuje změnu tlaku, která má fyzikální rozměr [Pa].

2.2. Struktura rozpoznávače hlasových příkazů

Na obrázku 2.2 vidíme blokové schéma systému pro rozpoznávání izolovaných hlasových příkazů online[2]. Zvuk je nejprve pomocí mikrofonu přeměněn na spojitý analogový signál. Protože veškeré další zpracování je prováděno digitálně, je analogový signál digitalizován, tedy proměněn na posloupnost diskrétních vzorků. Jak bylo řečeno v předcházející kapitole, výpočet evolučních charakteristik provádíme vždy na malém úseku signálu, na kterém je signál možno považovat za stacionární. Tento malý úsek signálu se nazývá *rámec*. Posloupnost vzorků je tedy rozdělena na rámce. V následujícím kroku jsou pro každý rámec vypočteny jeho *parametry*, což jsou vpodstatě okamžité hodnoty evolučních charakteristik. Tímto postupem byl tedy vstupní zvukový signál převeden na posloupnost parametrů.



Obrázek 2.2: Blokové schéma rozpoznávače hlasových příkazů

Další postup záleží na tom, zda se systém nachází v režimu *učení* nebo v režimu *rozpoznávání*, jak je na schématu naznačeno přepínačem. V režimu učení (horní tři polohy přepínače) uživatel říká jedno ze slov, které bude chtít později rozpoznávat. Vypočtené sekvence parametrů jsou přitom ukládány do jedné z pamětí, které jsou označeny jako *vzor 1*, *vzor 2*, *vzor 3*. Počet vzorů může být samozřejmě libovolný. V režimu rozpoznávání (dolní poloha přepínače) jsou pak tyto vzory čteny z paměti. Každý vzor je nezávisle porovnáván s posloupností parametrů, vypočtenou ze zvuku, který chceme rozpoznat. Po ukončení procesu srovnávání získáme informaci, jak výrazně se lišil rozpoznávaný zvuk od jednotlivých vzorů (hodnoty *rozdíl 1*, *rozdíl 2*, *rozdíl 3*). Pokud byla zjištěna pouze malá odlišnost mezi rozpoznávaným zvukem a některým ze vzorů, je nadřazenému systému předána informace o úspěšném detekování příkazu.

[2]Termínem *online* máme na mysli skutečnost, že přicházející zvuk je ihned zpracováván a informace o detekování příkazu je k dispozici okamžitě po ukončení zvukového vstupu.

Klíčovým krokem ve výše uvedeném postupu je jednak výpočet parametrů rámce, kde byly použity tzv. *Mel-frekvenční cepstrální koeficienty*, a dále pak srovnání rozpoznávané sekvence parametrů se vzorem, kde byla použita metoda *dynamického borcení času*. Celý výše uvedený postup bude podrobněji popsán v následujících kapitolách.

2.3. Snímání zvuku a digitalizace

Při použití běžného počítače, který je vybaven mikrofonem a zvukovou kartou, je snímání zvuku a jeho digitalizace triviální úlohou. Vpodstatě jedinou ovlivnitelnou hodnotou tohoto kroku je vzorkovací frekvence, o to pozorněji se ovšem musíme věnovat její volbě.

Při vzorkování spojitého signálu vzorkovací frekvencí f_S může výsledný diskrétní signál obsahovat pouze frekvenční složky z intervalu $\langle 0; f_N \rangle$, kde f_N je tzv. *Nyquistova frekvence*, která je určena Nyquist-Shannon-Kotelnikovým teorémem

$$f_N = f_S/2 \tag{2.1}$$

Pokud vstupní analogový signál obsahuje i jiné frekvenční složky, dojde k tzv. *aliasingu*, kdy se tyto vyšší frekvence určitým způsobem zobrazí do výše uvedeného intervalu $\langle 0; f_N \rangle$. Výstupní diskrétní signál pak již není správnou reprezentací původního signálu[3]. Proto obvykle před vzorkováním vstupní signál filtrujeme antialiasingovým filtrem. Jedná se o filtr typu dolní propust, který nepropouští frekvence vyšší než f_N. Zabráníme tedy aliasingu, ovšem o informaci obsaženou ve vyšších frekvenčních složkách nenávratně přijdeme. Z výše uvedeného plyne, že vzorkovací frekvenci musíme volit minimálně jako dvojnásobek nejvyšší frekvence, kterou chceme zachovat.

Shora pro vzorkovací frekvenci žádné teoretické omezení neexistuje. S rostoucí vzorkovací frekvencí ovšem lineárně narůstá i množství zpracovávaných vzorků. Pro účely testování algoritmu v MATLABu toto není příliš na závadu, ale v případě hlasového modulu vzorkovací frekvence přímo určuje potřebný výpočetní výkon a množství paměti. Proto provedeme volbu vzorkovací frekvence již nyní a stejnou vzorkovací frekvenci použijeme jak pro testování algoritmu v MATLABu, tak pro konečnou implementaci. Volbu provedeme na základě dvou nezávislých experimentů.

2.3.1. Volba f_S na základě výkonové spektrální hustoty

Výkonová spektrální hustota udává, jak výrazně jsou v signálu zastoupeny jednotlivé frekvenční složky. Na základě výkonové spektrální hustoty tedy můžeme zvolit vzorkovací frekvenci tak, abychom neodstranili žádné výrazné frekvenční složky řeči.

Uvažujme spojitý signál $x(t)$ a jeho diskrétní reprezentaci $x[n]$ o délce N vzorků, tedy $n \in \{0, 1, \ldots, N-1\}$. Tuto diskrétní reprezentaci určíme

$$x[n] = x(T.n) \tag{2.2}$$

[3]Pokud bychom se pokusili původní analogový signál rekonstruovat, mohli bychom obdržet obecně velmi odlišný signál.

kde $T = 1/f_S$ je perioda vzorkování. Pak diskrétní Fourierovu transformaci signálu spočteme jako

$$X[k] = \sum_{n=0}^{N-1} x[n].\mathrm{e}^{-2\pi \mathrm{j} nk/N} \tag{2.3}$$

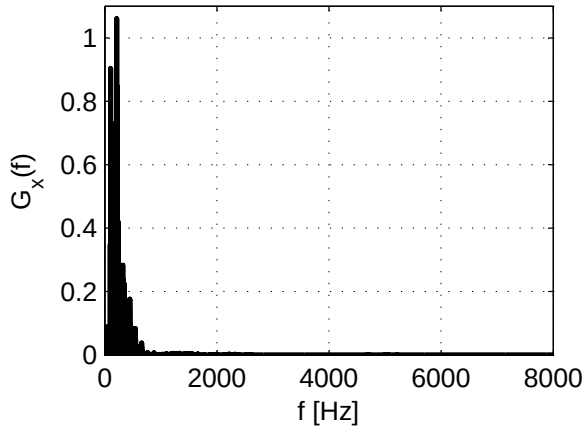
kde $k \in \{0, 1, \ldots, N-1\}$. Spektrální hustotu výkonu pak můžeme odhadnout pomocí vzorce

$$\hat{G}_x\left(k.\frac{f_S}{N}\right) = \frac{1}{N}|X[k]|^2 \tag{2.4}$$

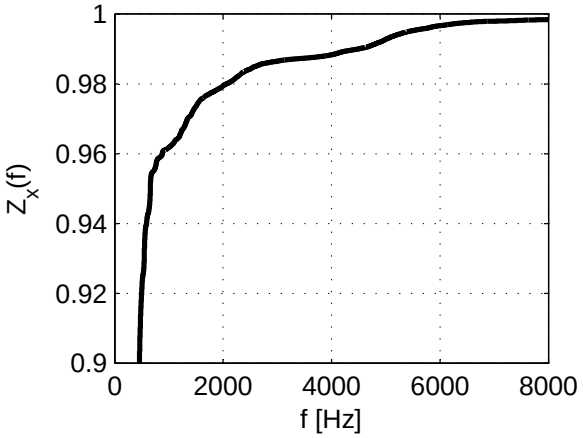
Odhad spektrální hustoty výkonu je symetrický okolo $k = N/2$, pokud je signál $x(t)$ reálný, což je v našem případě vždy splněno. Proto má nadále význam zabývat se pouze $k \in \{0, 1, \ldots, \lfloor N/2 \rfloor\}$.

Při praktickém provedení snímáme mikrofonem řečový signál dostatečné délky (aby byl výsledek reprezentativní) a zpracováváme jej pomocí počítače. Problém spočívá v tom, že při výpočtu výkonové spektrální hustoty pracujeme s diskrétním signálem, jehož frekvenční spektrum je shora omezeno. Proto použijeme v tomto experimentu vzorkovací frekvenci $f_S = 44\,100$ Hz, při které může diskrétní signál obsáhnout celé slyšitelné pásmo kmitočtů. Dále je nutno upozornit, že dle definice v kapitole 2.1 by měl mít signál $x(t)$ nulovou střední hodnotu. Vlivem nedokonalosti měřícího řetězce však pro $x(t)$, který získáme měřením, toto platit nemusí. Z tohoto důvodu uměle dodefinujeme $\hat{G}_x(0) = 0$.

Signál o délce 12 s, který byl použit pro tento experiment, je uložen v souboru `rec.wav`. Tento signál zpracujeme v MATLABu skriptem `shv.m`. Výsledný graf spektrální hustoty výkonu ukázkového signálu se nachází na obrázku 2.3. Je vykreslen pouze rozsah frekvencí do 8 kHz, neboť u vyšších frekvencí je spektrální hustota výkonu zanedbatelná.



Obrázek 2.3: Spektrální hustota výkonu



Obrázek 2.4: Poměrná přenesená energie

Pro zvýraznění malých hodnot se obvykle používá logaritmické zobrazení, za účelem volby vzorkovací frekvence však namísto toho raději definujme „*poměrnou přenesenou energii*“ vztahem

$$\hat{Z}_x\left(m.\frac{f_S}{N}\right) = \frac{\sum_{k=0}^{m} \hat{G}_x\left(k.\frac{f_S}{N}\right)}{\sum_{l=0}^{\lfloor N/2 \rfloor} \hat{G}_x\left(l.\frac{f_S}{N}\right)} \tag{2.5}$$

pro $m \in \{0, 1, \ldots, \lfloor N/2 \rfloor\}$. Tato veličina udává, jak velkou poměrnou část energie v signálu zachováme, pokud přenášené pásmo shora omezíme určitou frekvencí. Průběh, který

takto obdržíme, je zachycen na obrázku 2.4. Z důvodu rychlého růstu na počátku je zobrazena pouze zajímavá část okolo zlomu grafu. Z obrázku je možno odečíst, že 98 % výkonu signálu je obsaženo na kmitočtech do 2 kHz; tento bod leží přibližně v polovině „kolena“ grafu a lze jej tedy považovat za konec počátečního rychlého nárůstu. Pokud bychom chtěli zachovat 99 % výkonu, je nutno zpracovat kmitočty do 4,5 kHz; nad tuto hodnotu nemá smysl přenášené pásmo rozšiřovat, neboť spektrální hustota výkonu je zde již zanedbatelná. Z uvedených hodnot vyplývá dle (2.1) volba vzorkovací frekvence $f_S = 4 \ldots 9$ kHz.

2.3.2. Volba f_S na základě subjektivní srozumitelnosti

Tento způsob volby vzorkovací frekvence vychází z předpokladu, že řečový signál, který je dostatečně srozumitelný pro člověka, bude dobře zpracovatelný i automatickým rozpoznávačem. Postupujeme tedy tak, že omezujeme šířku pásma určitého řečového signálu pomocí filtru typu dolní propust a poslechem hodnotíme srozumitelnost signálu po průchodu filtrem. Šířku propustného pásma filtru postupně měníme, čímž získáme závislost subjektivní srozumitelnosti na šířce pásma. Opět použijeme signál ze souboru `rec.wav` a budeme jej zpracovávat pomocí skriptu `srozumitelnost.m`. Ve skriptu je použit eliptický IIR filtr se zvlněním propustné oblasti 0,5 dB, šířkou přechodové oblasti 100 Hz a útlumem v nepropustné oblasti 60 dB.

Výsledky při postupném zvětšování šířky pásma shrnuje tabulka 2.1. Za rozumnou minimální šířku pásma lze dle tabulky považovat hodnotu 2 kHz, kdy již lze jednotlivá slova rozpoznat bez nadměrného úsilí. Naopak pravděpodobně nemá význam zvyšovat šířku pásma nad 6 kHz, kde je již přínos minimální. Těmto skutečnostem odpovídá dle vztahu (2.1) volba vzorkovací frekvence $f_S = 4 \ldots 12$ kHz.

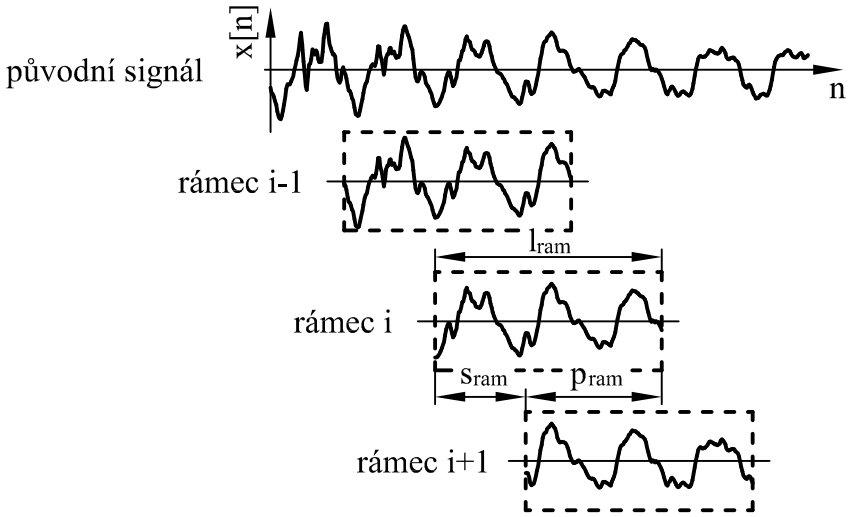
šířka pásma	subjektivní dojem
200 Hz	pouze tiché „bublání“; nelze poznat, zda jde o řeč či jiný zvuk
400 Hz	„obálka“ zvuku; již lze rozeznat, že se jedná o řeč, výjimečně lze rozeznat některá slova
600 Hz	asi polovinu slov lze rozeznat; hlasitost je již shodná s originálem
1 kHz	až na výjimky lze rozeznat všechna slova
1,5 kHz	začíná být rozeznatelné i zabarvení hlasu
2 kHz	rozeznávání slov je možné i bez soustředění
3 kHz	zabarvení hlasu se blíží skutečnosti
6 kHz	zabarvení hlasu plně odpovídá skutečnosti; začíná být patrný šum
8 kHz	zvýraznění sykavek
10 kHz	shodný s originálem (až na mírně menší obsah šumu)

Tabulka 2.1: Závislost srozumitelnosti na šířce pásma

Pro oba výše uvedené experimenty byl použit mužský hlas. Dá se předpokládat, že u ženského hlasu bude spektrální hustota výkonu posunuta k vyšším frekvencím a tím bude i potřebná vzorkovací frekvence vyšší. Zároveň je nutno podotknout, že vztah (2.1) platí pouze teoreticky a ve skutečnosti není možno celé frekvenční pásmo dokonale využít[4]. Proto volím (ve shodě s nepřímým doporučením [2]) vzorkovací frekvenci $f_S = 8$ kHz.

2.4. Dělení sekvence vzorků na rámce

Na obrázku 2.5 je zobrazen způsob, jakým jsou ze sekvence vzorků izolovány jednotlivé rámce. Vidíme, že tento proces je ovlivňován trojicí hodnot: *délkou rámce* l_{ram}, *posunem rámce* s_{ram} a *překrytím rámce* p_{ram}. Jedna z těchto hodnot je vždy jednoznačně určena volbou zbylých dvou hodnot.



Obrázek 2.5: Dělení signálu na rámce

Vzhledem k periodické struktuře řečového signálu by bylo vhodné, aby jediný rámec obsáhl více těchto period. Jinak by parametry, které z rámce později počítáme, měly víceméně náhodnou hodnotu, určenou tím, která část periodické struktury na daný rámec připadne. Délku rámce ovšem na druhé straně musíme volit dostatečně malou, aby se část signálu, kterou rámec obsahuje, dala považovat za stacionární. Pramen [2] uvádí jako obvykle používanou hodnotu $l_{ram} = 160 \ldots 200$ vzorků $= 20 \ldots 25$ ms při vzorkovací frekvenci $f_S = 8$ kHz.

Posun rámce musíme volit tak malý, abychom pomocí parametrů dostatečně popsali průběh nestacionarit v signálu. To vede na nutnost překrývání rámců, jak již bylo naznačeno na obrázku 2.5. Čím menší posun rámce zvolíme, tím více rámců získáme a tím rostou nároky na paměť i výpočetní výkon. Proto se dle pramene [2] obvykle používá kompromis $s_{ram} = 80$ vzorků $= 10$ ms.

[4]Pro plné využití teoretického frekvenčního pásma bychom potřebovali dokonalý antialiasingový filtr s nulovou šířkou přechodové oblasti, který není realizovatelný.

Nedostatkem postupu dle obrázku 2.5 je výrazná změna výkonové spektrální hustoty rámce oproti původnímu signálu. Tento efekt je nežádoucí, neboť dalším krokem je odhad parametrů rámce, který se zakládá v našem případě právě na výpočtu výkonové spektrální hustoty, jak bude uvedeno dále. Bez hlubší matematické analýzy uveďme, že řešení spočívá v aplikaci vhodného *okna* na rámec. Okno je vektor $w[n]$, kde $n \in \{0, 1, \ldots, l_{ram} - 1\}$. Aplikací okna pak máme na mysli vynásobení příslušného prvku rámce odpovídajícím prvkem okna. Vyhovující je například *Hammingovo okno*, které je definováno

$$w[n] = 0{,}54 - 0{,}46. \cos \frac{2\pi n}{l_{ram} - 1} \tag{2.6}$$

Toto okno tedy zmenší hodnotu okrajových prvků rámce, zatímco prvky uprostřed rámce ponechá takřka beze změny.

2.5. Mel-frekvenční cepstrální koeficienty

Mel-frekvenční cepstrální koeficienty (dále jen MFCC) představují jednu ze základních možností výpočtu parametrů řečového rámce. Jsou odvozeny z *cepstra*, které je pro spojitý signál $x(t)$ symbolicky definováno vztahem

$$C_x(t) = \mathcal{F}^{-1}\{\ln |\mathcal{F}\{x(t)\}|^2\} \tag{2.7}$$

Výpočet cepstra tedy spočívá v tom, že provedeme Fourierovu transformaci signálu, určíme druhou mocninu absolutní hodnoty, mezivýsledek zlogaritmujeme a pomocí inverzní Fourierovy transformace převedeme zpět do časové oblasti. Cepstrum se používá jako obecný nástroj pro analýzu signálů. Pomocí cepstra lze skupinu několika harmonických složek v původním signálu převést na jediné maximum v cepstru. Takto lze určit například opakovací periodu periodického signálu.

Motivace k využití cepstra při zpracování řeči je následující: vznik řeči si lze zjednodušeně představit jako průchod budícího signálu, který vzniká kmitáním hlasivek, přes ústní dutinu, která se chová jako filtr. Označíme-li budící signál $f(t)$ a impulzní odezvu filtru $h(t)$, pak lze výsledný signál $x(t)$ určit v časové oblasti pomocí *konvoluce*

$$x(t) = f(t) \star h(t) = \int_{-\infty}^{\infty} f(\tau).h(t - \tau).d\tau \tag{2.8}$$

Při zpracování řeči by bylo ideální zkoumat samostatně buzení $f(t)$ a filtr $h(t)$, máme ovšem k dispozici jen výsledný řečový signál $x(t)$. Snažíme se tedy o *dekonvoluci*. Zkusme dosadit vztah (2.8) do (2.7) a upravme:

$$\begin{aligned}
C_x(t) &= \mathcal{F}^{-1}\{\ln |\mathcal{F}\{f(t) \star h(t)\}|^2\} && (2.9)\\
C_x(t) &= \mathcal{F}^{-1}\{\ln |\mathcal{F}\{f(t)\}.\mathcal{F}\{h(t)\}|^2\} && (2.10)\\
C_x(t) &= \mathcal{F}^{-1}\{\ln |\mathcal{F}\{f(t)\}|^2 + \ln |\mathcal{F}\{h(t)\}|^2\} && (2.11)\\
C_x(t) &= \mathcal{F}^{-1}\{\ln |\mathcal{F}\{f(t)\}|^2\} + \mathcal{F}^{-1}\{\ln |\mathcal{F}\{h(t)\}|^2\} && (2.12)\\
C_x(t) &= C_f(t) + C_h(t) && (2.13)
\end{aligned}$$

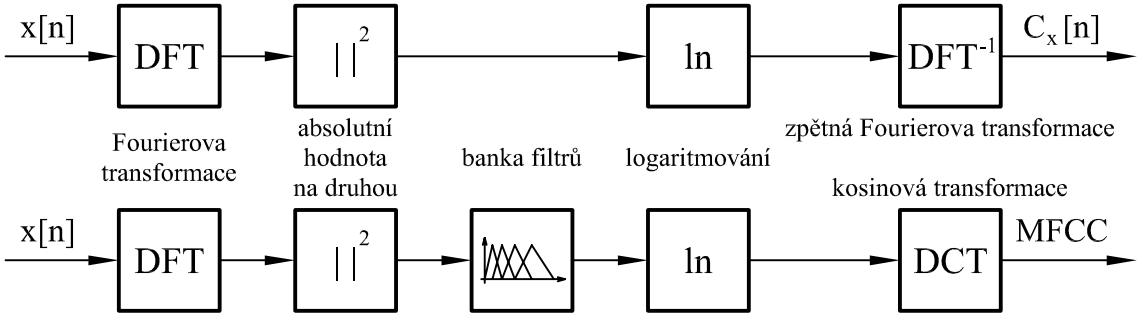
Vidíme tedy, že konvoluce v časové oblasti se změnila na součin ve frekvenční oblasti, který byl logaritmem převeden na součet. Díky linearitě inverzní Fourierovy transformace pak je cepstrum výsledného signálu $C_x(t)$ pouhým součtem cepstra buzení $C_f(t)$ a cepstra impulzní odezvy filtru $C_h(t)$. Protože buzení a impulzní odezva filtru jsou obvykle rozdílné povahy, lze pak v cepstru řečového signálu $C_x(t)$ oba vlivy izolovat. V praxi toto nefunguje zcela dokonale, přesto je možno výše uvedenou vlastnost cepstra využít.

Při praktickém výpočtu cepstra pracujeme s diskrétním signálem $x[n]$. Definice cepstra diskrétního signálu je analogická vztahu (2.7), pouze musíme použít diskrétní Fourierovu transformaci (2.3). Povšimněme si také toho, že výpočet cepstra v sobě zahrnuje výpočet spektrální hustoty výkonu (2.4). Výpočet MFCC rámce se liší od výpočtu cepstra ve dvou ohledech, jak je znázorněno na obrázku 2.6:

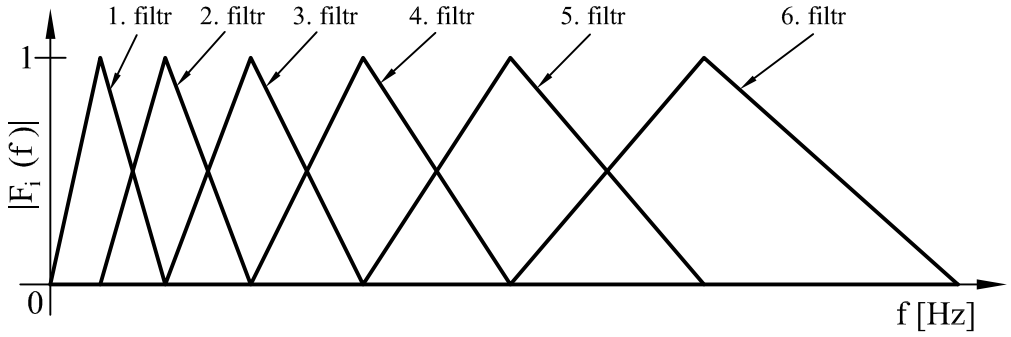
1. Před logaritmováním dochází k transformaci frekvenční osy. Při výpočtu diskrétní Fourierovy transformace signálu obdržíme na všech frekvencích stejné „frekvenční rozlišení“ (tj. frekvence, ke kterým náleží sousední prvky vypočteného vektoru, se všude liší o stejnou hodnotu). Naopak lidský sluch (k němuž se při konstrukci rozpoznávače snažíme přiblížit) má větší „frekvenční rozlišení“ na nižších frekvencích než na vyšších, tuto skutečnost postihuje jednotka kmitočtu zvaná Mel. Přepočet z Hertzů na Mely je definován vztahem

$$f[\mathrm{Mel}] = 2959.\log_{10}\left(1 + \frac{f[\mathrm{Hz}]}{700}\right) \tag{2.14}$$

   Transformaci frekvenční osy provedeme pomocí *banky filtrů*. Na frekvenční osu v Melech rozmístíme rovnoměrně filtry s amplitudovými chrarakteristikami $|F_i(f)|$. Obvykle volíme trojůhelníkový tvar charakteristik, přičemž jednotlivé charakteristiky se překrývají. Tyto filtry budou díky vztahu (2.14) na frekvenční ose v Hertzích rozmístěny nerovnoměrně, jak je znázorněno na obrázku 2.7. Výstupem transformace frekvenční osy je energie na výstupu těchto filtrů. Protože ale máme z předešlého kroku k dispozici odhad výkonové spektrální hustoty vstupního signálu, nemusíme počítat odezvu filtrů na vstupní signál; postačí pouze provést skalární součin odhadu výkonové spektrální hustoty s vektorem, popisujícím amplitudovou charakteristikou příslušného filtru. Počet filtrů vhodně volíme, například v prameni [2] je použito 23 filtrů.



Obrázek 2.6: Výpočet cepstra a MFCC z diskrétního signálu



Obrázek 2.7: Banka filtrů

2. Namísto diskrétní inverzní Fourierovy transformace použijeme kosinovou transformaci, která je definována

$$X_{DCT}[k] = w[k].\sum_{n=0}^{N-1} x[n].\cos\frac{\pi k(2n+1)}{2N} \tag{2.15}$$

$$w[k] = \begin{cases} \frac{1}{\sqrt{N}} & \text{pro } k = 0 \\ \sqrt{\frac{2}{N}} & \text{pro } k \neq 0 \end{cases} \tag{2.16}$$

Vzhledem k tomu, že kosinová transformace je odvozena z Fourierovy transformace, mají tyto dvě transformace podobné vlastnosti a náhrada je možná. Díky použití kosinové transformace obdržíme na výstupu reálná čísla, na rozdíl od Fourierovy transformace, jejíž výstup je komplexní.

Dodejme, že pro rozpoznávání řeči obvykle využíváme jen několik prvních MFCC a zbylé v dalších výpočtech neuvažujeme. V prameni [2] je použito prvních 13 MFCC.

2.6. Dynamické borcení času

V předcházejících krocích jsme převedli řečový signál (který chceme rozpoznat) na sekvenci vektorů, přičemž každý vektor reprezentuje jeden rámec a obsahuje MFCC tohoto rámce. Tyto vektory označme $\mathbf{o}(t)$, kde $t \in \{1, 2, \ldots, T\}$ a celou sekvenci vektorů označme $\mathbf{O}$. Dále předpokládejme, že máme k dispozici obdobnou sekvenci vektorů, která odpovídá určitému vzoru. Tato sekvence obsahuje vektory $\mathbf{v}(r)$, kde $r \in \{1, 2, \ldots, R\}$; celou sekvenci pak označme jako $\mathbf{V}$. Pokud chceme zjistit, zda je rozpoznávaný řečový signál podobný vzoru či nikoliv, musíme nějakým způsobem ohodnotit podobnost či rozdílnost sekvencí $\mathbf{O}$ a $\mathbf{V}$.

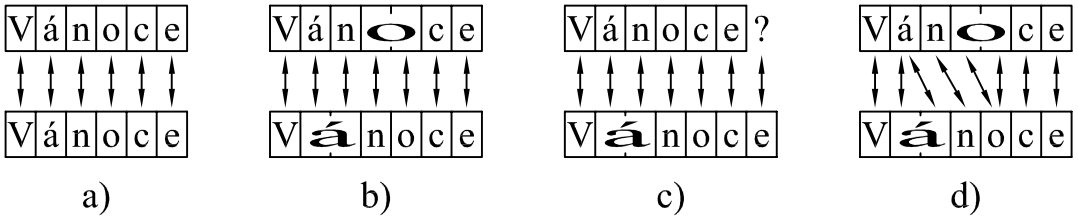
Začněme řádově jednodušší úlohou: pokud chceme ohodnotit rozdílnost jediného vektoru $\mathbf{o}(t)$ od jediného vektoru $\mathbf{v}(r)$, můžeme snadno použít např. běžnou Euklidovskou metriku. Pokud prvky vektoru $\mathbf{o}(t)$ označíme $o(i)$ a prvky vektoru $\mathbf{v}(r)$ označíme $v(i)$, přičemž $i \in \{1, 2, \ldots, I\}$, pak lze rozdílnost zmíněných vektorů definovat

$$d\Big(\mathbf{o}(t), \mathbf{v}(r)\Big) = \sqrt{\sum_{i=1}^{I} [o(i) - v(i)]^2} \tag{2.17}$$

Pokud chceme srovnat dvě sekvence vektorů $\mathbf{O}$ a $\mathbf{V}$, mohli bychom např. sečíst rozdílnosti vektorů se stejným pořaďovým číslem, tedy

$$d(\mathbf{O},\mathbf{V}) = \frac{1}{T}\sum_{t=1}^{T} d\Big(\mathbf{o}(t),\mathbf{v}(t)\Big) \quad \text{za podmínky } T = R \tag{2.18}$$

Člen $1/T$ zajišťuje, že srovnání bude nezávislé na délce sekvencí (tj. nebudeme zvýhodňovat krátké sekvence před dlouhými). Srovnání dle (2.18) vyžaduje, aby oba řečové signály byly přesně „zarovnány“. Popsaná situace je znázorněna na obrázku 2.8 a), přičemž sekvence vektorů $\mathbf{O}$ a $\mathbf{V}$ byly pro názornost nahrazeny hláskami slova „Vánoce“. V teoretickém případě a) by bylo srovnání provedeno správně. V praxi ovšem člověk nemluví nikdy s naprosto stejným časováním, takže srovnání v případě b), kdy je střed slova posunut, by nevedlo na správný výsledek. V případě c), kdy mají srovnávaná slova rozdílnou délku, by uvedený postup vůbec nebylo možno použít (poslední hlásku druhého slova není s čím srovnat). Ideální by bylo počítat rozdílnost těch vektorů $\mathbf{o}(t)$ a $\mathbf{v}(r)$, které patří ke stejným hláskám ve slově, jak je naznačeno na obrázku 2.8 d). Tento přístup používá právě *dynamické borcení času* (dále jen DTW).



Obrázek 2.8: Různé případy při porovnávání sekvencí

Vysvětlení funkce DTW začněme tím, že symboly t a r (které jsme dosud považovali za postupně rostoucí indexy) nahradíme funkcemi $t(k)$ a $r(k)$, kde k představuje *krok srovnávání*. Musí platit

$$\left.\begin{array}{l} t(k) \in \{1,2,\ldots,T\} \\ r(k) \in \{1,2,\ldots,R\} \end{array}\right\} \forall k \in \{1,2,\ldots,K\} \tag{2.19}$$

Je zřejmé, že pomocí funkcí $t(k)$ a $r(k)$ můžeme ze sekvencí $\mathbf{O}$ a $\mathbf{V}$ vybrat libovolné vektory $\mathbf{o}\Big(t(k)\Big)$ a $\mathbf{v}\Big(r(k)\Big)$ pro k-tý krok srovnávání. Při vhodné volbě funkcí $t(k)$ a $r(k)$ tak můžeme docílit efektu, znázorněného na obrázku 2.8 d). Rozdílnost sekvencí $\mathbf{O}$ a $\mathbf{V}$ pak nebudeme určovat pomocí vztahu (2.18), ale takto:

$$d(\mathbf{O},\mathbf{V}) = \frac{1}{T+R}\sum_{k=1}^{K} w(k).d\Big(\mathbf{o}\Big(t(k)\Big),\mathbf{v}\Big(r(k)\Big)\Big) \tag{2.20}$$

Váha k-tého kroku srovnání $w(k)$ bude objasněna později. Člen $1/(T+R)$ zajistí, aby výsledek byl opět nezávislý na délce srovnávaných sekvencí.

Vzhledem k tomu, že je možno vytvořit celkem $(T.R)^K$ různých dvojic funkcí $t(k)$ a $r(k)$, definujme nyní na tyto funkce další omezení[5]:

1. Srovnání by mělo začínat na začátku obou sekvencí a končit na konci obou sekvencí, tedy $t(1) = 1$, $r(1) = 1$, $t(K) = T$, $r(K) = R$.

2. Obě funkce by měly být neklesající, tedy $t(k) - t(k-1) \geq 0$, $r(k) - r(k-1) \geq 0$. To znamená, že ani v jedné sekvenci vektorů se nesmíme vracet zpět.

3. Nemá smysl opakovaně porovnávat stejnou dvojici vektorů, alespoň jedna z funkcí se musí oproti předešlému kroku změnit, tedy $\Big(t(k) \neq t(k-1)\Big) \vee \Big(r(k) \neq r(k-1)\Big)$.

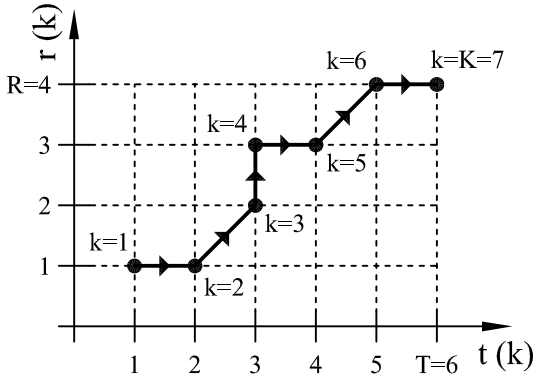
4. Je vhodné, abychom žádný vektor ze sekvence vektorů nepřeskočili. Můžeme se posouvat nejvýše o jeden vektor, tedy $t(k) - t(k-1) \leq 1$, $r(k) - r(k-1) \leq 1$.

Chápeme-li funkce $t(k)$ a $r(k)$ jako souřadnice v rovině $\{1, 2, \ldots, T\} \times \{1, 2, \ldots, R\}$, je možno průběh srovnávání vykreslit pomocí *cesty* mezi body této roviny. Obrázek 2.9 ukazuje příklad cesty, která respektuje výše uvedená omezení. Do každého bodu roviny $[t, r]$ (s výjimkou bodů, kde $(t = 1) \vee (r = 1)$) se můžeme dostat třemi způsoby:

1. z bodu $[t-1, r]$, tj. posunem v rozpoznávané sekvenci

2. z bodu $[t-1, r-1]$, tj. posunem v obou sekvencích

3. z bodu $[t, r-1]$, tj. posunem ve vzorové sekvenci



Obrázek 2.9: Příklad cesty, popisující průběh srovnávání

Nyní musíme určit, kterou cestu z množiny možných cest použijeme pro srovnání. Jak bylo již dříve uvedeno, je našim cílem srovnávat dvojice **o** a **v**, které jsou si co nejvíce podobné, tedy jejich rozdílnost je malá. Pak bude i celková rozdílnost sekvencí **O** a **V** malá. Řešením by tedy mohlo být postupné vygenerování všech cest a výpočet rozdílnosti sekvencí pro každou z těchto cest (dále jen *cena cesty*). Za optimální lze pak považovat takovou cestu, která generuje minimální cenu cesty.

[5]Tato omezení je možno dle potřeby a povahy srovnávaných sekvencí definovat i jinak.

Existuje však méně náročný postup: pro každý bod roviny určíme cenu optimální cesty od počátku až do tohoto bodu. A protože do každého bodu roviny se lze dostat ze třech jiných bodů, můžeme cenu optimální cesty od počátku do bodu $[t, r]$ určit jako

$$g(t,r) = \min \left\{ \begin{array}{c} g(t-1,r) + d\Big(\mathbf{o}(t), \mathbf{v}(r)\Big) \\ g(t-1,r-1) + 2.d\Big(\mathbf{o}(t), \mathbf{v}(r)\Big) \\ g(t,r-1) + d\Big(\mathbf{o}(t), \mathbf{v}(r)\Big) \end{array} \right\} \quad \text{pro } t \neq 1, r \neq 1 \tag{2.21}$$

U prostřední možnosti násobíme rozdílnost vektorů dvěma, neboť pohyb „po úhlopříčce“ vede na kratší cestu (menší K) než pohyb v jedné souřadnici; tímto opatřením zajistíme rovnocennost všech pohybů. Uvedený vztah je vlastně rekurzivní formou zápisu vztahu (2.20) a zmiňovaná dvojka představuje člen $w(k)$ z tohoto vztahu. Pro nastartování výpočtu je třeba dodefinovat

$$g(1,r) = g(1,r-1) + d\Big(\mathbf{o}(1), \mathbf{v}(r)\Big) \quad \text{pro } r \neq 1 \tag{2.22}$$

$$g(t,1) = g(t-1,1) + d\Big(\mathbf{o}(t), \mathbf{v}(1)\Big) \quad \text{pro } t \neq 1 \tag{2.23}$$

$$g(1,1) = d\Big(\mathbf{o}(1), \mathbf{v}(1)\Big) \tag{2.24}$$

Po ohodnocení všech bodů roviny získáme v $g(T, R)$ cenu optimální cesty přes obě celé sekvence, a tedy

$$d(\mathbf{O}, \mathbf{V}) = \frac{g(T,R)}{T+R} \tag{2.25}$$

2.7. Testování algoritmu

Funkčnost metody pro rozpoznávání hlasových příkladů byla před započetím vývoje samotného modulu otestována v programu MATLAB. Přestože cílem této práce je vývoj online rozpoznávače, v MATLABu byl naprogramován offline rozpoznávač, který vstupní signál načítá z předem připravených souborů. Díky tomu probíhalo ladění vždy se stejnými daty a bylo tudíž jednodušší.

Pro testování algoritmu bylo použito pět vzorů. Jednalo se o slova „doleva“, „doprava“, „vpřed“, „vzad“ a „stůj“. Tyto vzory jsou uloženy v souborech `vzor_1.wav` až `vzor_5.wav`. Dále byla vytvořena sada 200 slov, určených k rozpoznání (ke každému vzoru patří 40 slov ze sady). Tato slova jsou uložena v souborech `test_1_1.wav` až `test_5_40.wav`, přičemž první číslo udává, ke kterému vzoru dané slovo patří (tato informace slouží pouze k závěrečnému zhodnocení úspěšnosti rozpoznání) a druhé číslo pouze odlišuje jednotlivá slova, patřící ke společnému vzoru.

Ke zpracování výše uvedených souborů slouží skript `test.m`. Nejprve se postupně načítají soubory se vzory, které jsou převedeny na sekvence vektorů. Pak jsou načítána jednotlivá slova, určená k rozpoznání. Po převodu na sekvenci vektorů je každé slovo pomocí DTW porovnáno postupně se všemi vzory. Následně je slovo přiřazeno k tomu vzoru, u kterého byla pomocí DTW vypočtena nejmenší odlišnost mezi slovem a vzorem. Na závěr je vyhodnocen počet chybných rozpoznání, chybovost v procentech a délka trvání rozpoznávání všech slov.

Skript `test.m` používá několik dílčích funkcí:

- `dtw.m` zajišťuje srovnání dvou sekvencí vektorů pomocí DTW
- `filtry.m` slouží k návrhu banky filtrů
- `hz2mel.m` převádí Hertze na Mely dle vztahu (2.14)
- `mfcc.m` převede vstupní signál na sekvenci vektorů, které obsahují MFCC
- `rozdel.m` zajišťuje dělení vstupního signálu na rámce

Proces rozpoznávání je možno ovlivnit pomocí parametrů, které se nastavují na začátku souboru `test.m` a které jsou uvedeny v tabulce 2.2. Výchozí hodnoty těchto parametrů byly převzaty z pramene [2]. Vzorkovací frekvence a posun rámce byly ponechány beze změny, délka rámce však byla změněna z původních 180 vzorků na 128 vzorků. Z důvodů, které budou objasněny v kapitole 4.4.2 je totiž vhodné, abychom Fourierovu transformaci prováděli na vektoru o délce 2^n prvků, kde n je přirozené číslo. Při délce rámce 180 vzorků tedy musíme rámec doplnit 76 nulovými prvky a provést 256-bodovou Fourierovu transformaci. Pokud délku rámce omezíme na 128 vzorků, můžeme přímo použít 128-bodovou Fourierovu transformaci a tím zmenšit zejména paměťové nároky rozpoznávače. To může být užitečné při konečné implementaci rozpoznávače. Zkrácení rámce nezhoršilo úspěšnost rozpoznávání. Dále bylo během testování zjištěno, že je taktéž možno snížit počet využívaných MFCC a odpovídajícím způsobem zmenšit počet filtrů v bance filtrů, což opět vede k výrazným úsporám paměti.

parametr	výchozí hodnota (dle [2])	konečná hodnota (použita dále)
vzorkovací frekvence	8 kHz	8 kHz
délka rámce	180 vzorků	128 vzorků
posun rámce	80 vzorků	80 vzorků
počet filtrů v bance	23	12
počet využitých MFCC	13	5

Tabulka 2.2: Parametry algoritmu

I s výše popsanými úpravami hodnot parametrů se rozpoznávač dopustil na sadě testovacích dat (200 slov) pouze jediné chyby, úspěšnost rozpoznání tedy dosahuje 99,5%. Tím byla prakticky ověřena použitelnost metody. Je ovšem nutno vzít v potaz, že vzory i slova k rozeznávání byly zaznamenány v poměrně krátkém časovém rozpětí, navíc v prostředí prakticky bez rušivých zvuků. Při nasazení rozpoznávače v reálných podmínkách lze očekávat výrazně nižší úspěšnost.

3. Návrh hardware

Ústřední součástí hardware hlasového modulu je mikrokontrolér, který zajišťuje vykonávání algoritmu, popsaného v předcházející části, a obsluhu zbylých částí modulu. Návrh tedy začneme výběrem tohoto mikrokontroléru, neboť další postup je na této volbě úzce závislý.

3.1. Výběr mikrokontroléru

Z požadavku na zpracování vstupního zvukového signálu v reálném čase plyne potřeba poměrně velkého výpočetního výkonu mikrokontroléru. Z charakteru algoritmu plyne nutnost provádění výpočtů s přesností 16 bitů nebo větší, je tedy vhodné použít minimálně 16-bitový mikrokontrolér. Těmto požadavkům vyhovuje například řada signálových kontrolérů 56800 od firmy Freescale. Termín *signálový kontrolér* značí, že tyto obvody v sobě kombinují výkonné jádro (typické pro signálové procesory) a širokou škálu periferií (typické pro mikrokontroléry). Ačkoliv tyto obvody jsou primárně určeny pro řízení pohonů a výkonovou elektroniku, lze je stejně dobře použít i pro obecné zpracování signálu. Řadu 56800 volím zejména proto, že s ní již mám určité zkušenosti z dřívější doby.

Z řady 56800 dále volím konkrétní typ 56F805, jehož hlavní vlastnosti shrnuje tabulka 3.1. Za povšimnutí stojí poměrně malé množství datové paměti RAM, které může být limitujícím faktorem při implementaci algoritmu. Toto omezení by bylo možno odstranit připojením externí paměti, což by ovšem zkomplikovalo návrh desky plošných spojů a proto tato možnost není využita. Z periferií stojí za zmínku dvojice AD převodníků, které lze použít ke zpracování signálu z mikrofonu. Rozhraní pro sériovou komunikaci lze využít ke snadnému připojení ke sběrnici RS-485, přes kterou modul komunikuje s nadřazeným systémem. Podrobný popis periferií lze nalézt v [4].

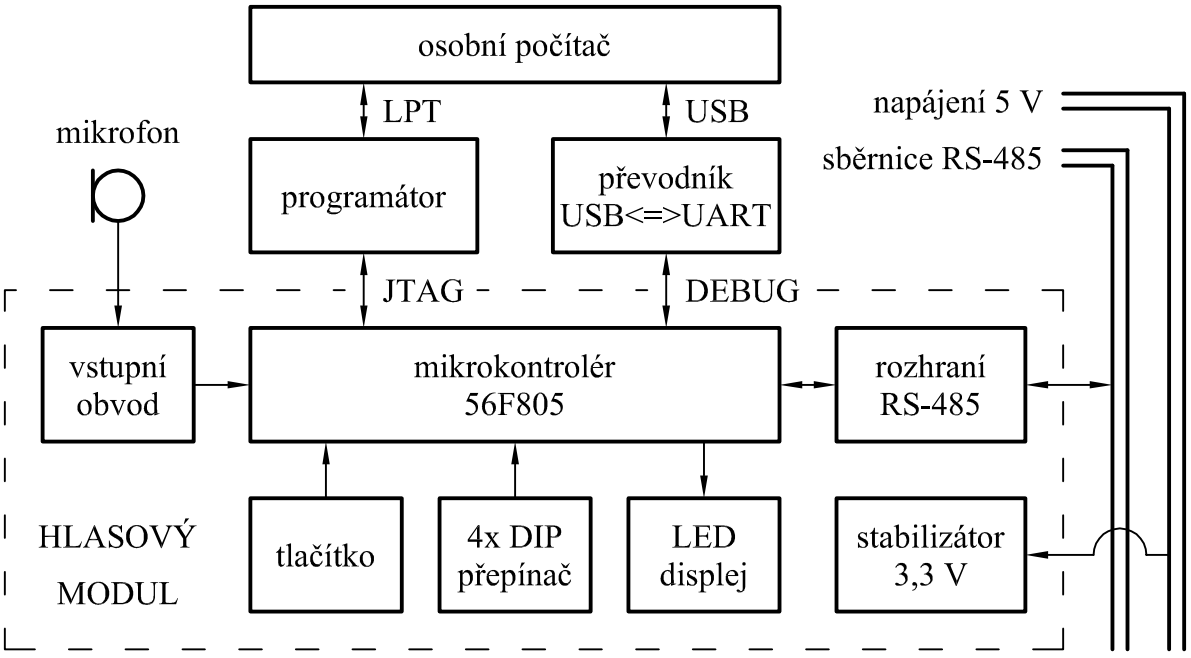
výpočetní výkon	40 MIPS při frekvenci jádra 80 MHz
programová paměť	63 kB FLASH + 1 kB RAM
datová paměť	8 kB FLASH + 4 kB RAM
periferie	dva 6-kanálové generátory PWM
	dva 4-kanálové, 12-bitové AD převodníky
	dva dekodéry kvadraturního signálu
	rozhraní sběrnice CAN
	dvě rozhraní pro sériovou komunikaci
	rozhraní SPI
	čtyři čtyřnásobné čítače
programovací rozhraní	JTAG/OnCE
napájení	3,3 V, max. 152 mA
pouzdro	LQFP, 144 vývodů

Tabulka 3.1: Vlastnosti signálového kontroléru 56F805 (převzato z [3])

Pouzdro LQFP se 144 vývody se může zdát pro danou aplikaci zbytečně velké (většina vývodů zůstane nevyužita). Při použití mikrokontroléru s menším počtem vývodů by ovšem muselo být kolem mikrokontroléru volné místo, sloužící k přeskupení vodivých cest, a celková plocha desky plošných spojů by byla prakticky stejná. Vlastnosti výpočetního jádra souvisí spíše s návrhem software a budou proto popsány až v kapitole 4.1.

3.2. Blokové schéma hlasového modulu

Na obrázku 3.1 vidíme blokové schéma hlasového modulu. Signál z mikrofonu je zpracován ve vstupním obvodu a v mikrokontroléru. Mikrokontrolér komunikuje s nadřazeným systémem po sběrnici RS-485 skrze příslušné rozhraní. Hlasový modul je napájen z napětí 5 V, které je lineárním stabilizátorem sníženo na 3,3 V. Mikrokontrolér je možno programovat přes rozhraní JTAG, na které je připojen programátor. Programátor je připojen k osobnímu počítači pomocí paralelního portu (označen LPT). Zbylé prvky slouží pro usnadnění ladění hlasového modulu; jedná se o tlačítko, přepínače, sedmisegmentový displej s LED a ladící port (označen DEBUG). Ladící port slouží k obousměrné komunikaci s osobním počítačem. K USB portu počítače je ladící port připojen přes převodník (s integrovaným obvodem FT232RL či obdobným).



Obrázek 3.1: Blokové schéma hlasového modulu

V dalších kapitolách je uveden popis vstupního obvodu a rozhraní sběrnice RS-485. Zapojení tlačítka, displeje, stabilizátoru atd. je triviální a proto je uvedeno pouze v celkovém schématu, které je součástí přílohy.

3.3. Vstupní obvod

3.3.1. Koncepce vstupního obvodu

Vstupem hlasového modulu bude analogový signál z mikrofonu, který je potřeba převést do digitální podoby. Pro účely rozpoznávání hlasových příkazů počítáme s použitím elektretového mikrofonu, neboť je malý a levný. Úpravu signálu z mikrofonu a jeho digitalizaci lze řešit v zásadě dvěma způsoby:

1. Ke zpracování signálu z mikrofonu můžeme použít jednoúčelový integrovaný obvod, např. MAX9867 [5]. Obvody tohoto typu (nazývané obvykle kodeky) umožňují digitalizaci audiosignálu i opačný, tj. digitálně-analogový převod. Obsahují obvykle předzesilovače, výstupní zesilovače, filtry i 16-bitové převodníky, takže pracují prakticky bez vnějších součástek.

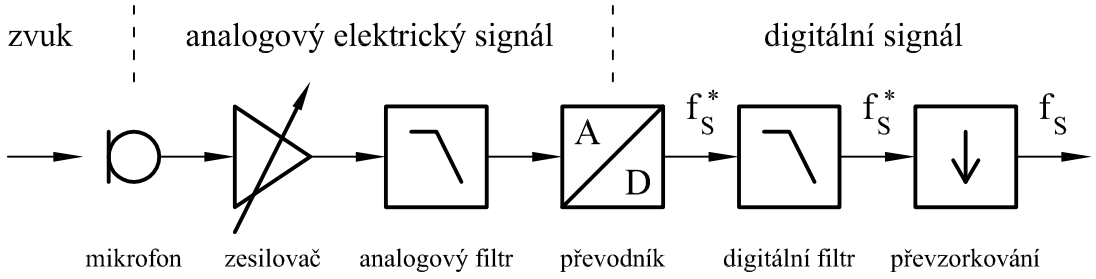
2. K provedení analogově-digitálního převodu můžeme použít 12-bitový AD převodník, který je součástí mikrokontroléru. Převodník musíme na vstupu doplnit předzesilovačem a antialiasingovým filtrem.

První možnost představuje elegantní hotové řešení, ovšem obvody tohoto typu jsou v kusovém množství špatně dostupné a vyžadují poměrně komplikovanou konfiguraci přes některý typ sériové sběrnice. Z těchto důvodů volím druhou variantu, bude použit vnitřní AD převodník mikrokontroléru, doplněný o předzesilovač a antialiasingový filtr.

Při návrhu antialiasingového filtru vycházíme ze vzorkovací frekvence $f_S = 8$ kHz, kterou jsme zvolili v závěru kapitoly 2.3.2, a také z odpovídající teoretické šířky pásma $f_N = 4$ kHz (dle vztahu (2.1)). Ideální antialiasingový filtr by propustil všechny frekvence pod 4 kHz a naopak všechny frekvence nad 4 kHz by zcela potlačil. Takový filtr není realizovatelný, zvolíme si tedy následující specifikace filtru:

- filtr propustí všechny frekvence pod 3,6 kHz
- zvlnění propustné části charakteristiky bude nejvýše 0,5 dB
- filtr potlačí všechny frekvence nad 4,4 kHz
- potlačení frekvencí v nepropustné části charakteristiky bude nejméně 60 dB

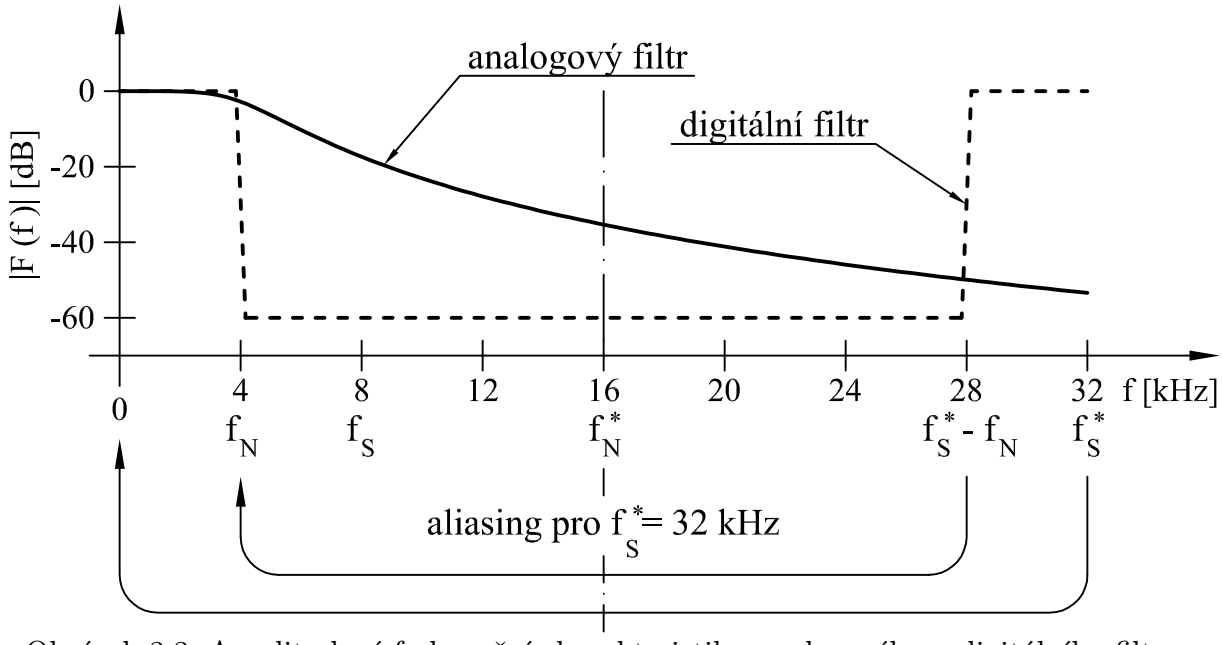
Navrhujeme-li filtr jako eliptický (tj. nejmenšího možného řádu), pak pro dodržení těchto specifikací potřebujeme filtr 7. řádu, jak je ukázáno ve skriptu `aafiltr.m`. Takový filtr je sice v analogové podobě realizovatelný, ale vyžadoval by poměrně velké množství přesných součástek. Proto volíme úplně odlišný přístup, který je znázorněn na obrázku 3.2: signál vzorkujeme se vzorkovací frekvencí f_S^*, která je několikanásobkem původní vzorkovací frekvence $f_S = 8$ kHz (díky tomu stačí při zachování původní šířky pásma analogový antialiasingový filtr s menší strmostí a tedy i nižšího řádu). Navzorkovaný signál je pak v mikrokontroléru zpracován digitálním filtrem (který má potřebnou strmost, ovšem jeho řád již není rozhodující) a jeho výstup převzorkujeme na požadovanou vzorkovací frekvenci f_S.



Obrázek 3.2: Principiální blokové schéma vstupní části

Vzorkovací frekvenci f_S^* volíme tak, aby analogový antialiasingový filtr byl snadno realizovatelný a přitom f_S^* nebyla příliš vysoká (s rostoucí f_S^* roste i výpočetní náročnost digitálního filtru). Z tohoto hlediska lze za optimum považovat $f_S^* = 4f_S = 32$ kHz, kdy vystačíme s analogovým filtrem 3. řádu. Použijeme filtr o lomovém kmitočtu 4 kHz s aproximací podle Butterwortha, který bude podrobně popsán v kapitole 3.3.2.

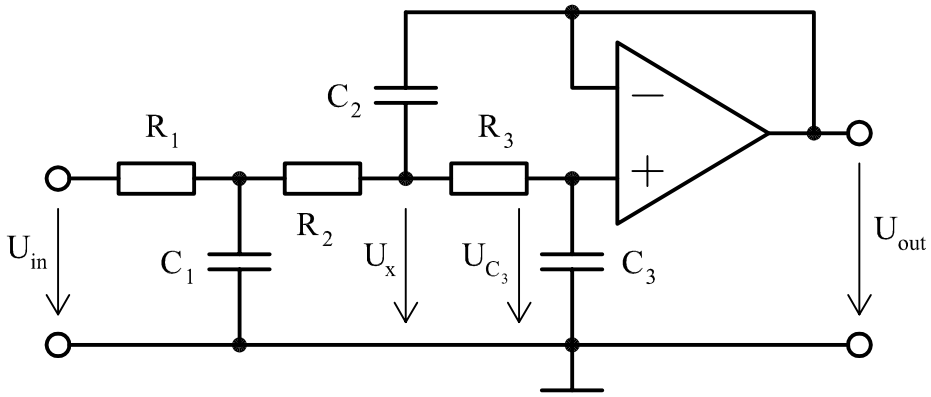
Rozbor uvedeného řešení ve frekvenční oblasti ukazuje obrázek 3.3, kde jsou vykresleny amplitudové charakteristiky analogového i digitálního filtru (charakteristika digitálního filtru je složitější, ale zobrazená aproximace pro demonstraci principu postačuje). Je zřejmé, že frekvence z intervalu $\langle 0; f_N)$ projdou oběma filtry prakticky nezměněny, což je žádoucí. Frekvence v intervalu $(f_N; f_N^*\rangle$ jsou výrazně potlačeny digitálním filtrem a částečně i analogovým filtrem. Frekvence v intervalu $\langle f_N^*; f_S^* - f_N)$ se díky aliasingu během vzorkování promítnou do intervalu $(f_N; f_N^*\rangle$ a jsou tedy také potlačeny digitálním filtrem. Teprve frekvence nad $f_S^* - f_N = 28$ kHz by mohly projít digitálním filtrem a znehodnotit užitečné pásmo $\langle 0; f_N)$, jsou však potlačeny analogovým filtrem, který má pro tyto frekvence útlum minimálně 50 dB. Tento rozbor dokazuje, že kombinací jednoduchého analogového filtru a digitálního filtru (který popíšeme v kapitole 4.3.1) lze dosáhnout podobného efektu, jako náročným analogovým filtrem 7. řádu.



Obrázek 3.3: Amplitudové frekvenční charakteristiky analogového a digitálního filtru

3.3.2. Návrh analogového filtru

Analogový filtr 3. řádu s aproximací dle Butterwortha lze realizovat pomocí řady různých zapojení. Zvoleno bylo řešení s operačním zesilovačem dle obrázku 3.4, neboť toto zapojení nevyžaduje použití cívek. Zapojení je převzato z [6]. Tento pramen uvádí i vzorce pro stanovení hodnot jednotlivých součástek. Vypočtené hodnoty součástek ovšem musíme v praxi zaokrouhlit na hodnoty z řady E12 (u odporů)[1] nebo E6 (u kondenzátorů)[2], čímž může dojít k nežádoucímu ovlivnění přenosové charakteristiky filtru. Abychom tento efekt minimalizovali, odvodíme analyticky přenos tohoto filtru a vybereme nejvhodnější kombinaci hodnot součástek.



Obrázek 3.4: Schéma zapojení filtru

Odvození provedeme s pomocí Laplaceovy transformace, která nahrazuje derivaci operátorem p. Při odvození použijeme vztahy pro impedanci rezistoru R a kondenzátoru C

$$Z_R = R \tag{3.1}$$

$$Z_C = \frac{1}{pC} \tag{3.2}$$

dále vzorec pro paralelní kombinaci impedancí Z_1 a Z_2

$$Z_1 \| Z_2 = \frac{Z_1 Z_2}{Z_1 + Z_2} \tag{3.3}$$

a vzorec pro výpočet výstupního napětí děliče, který je složen z impedancí Z_1 a Z_2 a je napájen napětím U

$$U_{Z_2} = U \frac{Z_2}{Z_1 + Z_2} \tag{3.4}$$

Do sítě pasivních součástek vstupují napětí U_{in} a U_{out}, pro odvození přenosu filtru potřebujeme nejprve určit napětí U_x. Použijeme proto princip superpozice a napětí U_x stanovíme jako součet „příspěvků“ od napětí U_{in} a U_{out}:

$$U_x = U_x^{U_{in}} + U_x^{U_{out}} \tag{3.5}$$

[1] Řada E12 obsahuje hodnoty ... 1; 1,2; 1,5; 1,8; 2,2; 2,7; 3,3; 3,9; 4,7; 5,6; 6,8; 8,2; 10; 12; 15 ... atd.
[2] Řada E6 obsahuje hodnoty ... 1; 1,5; 2,2; 3,3; 4,7; 6,8; 10; 15 ... atd.

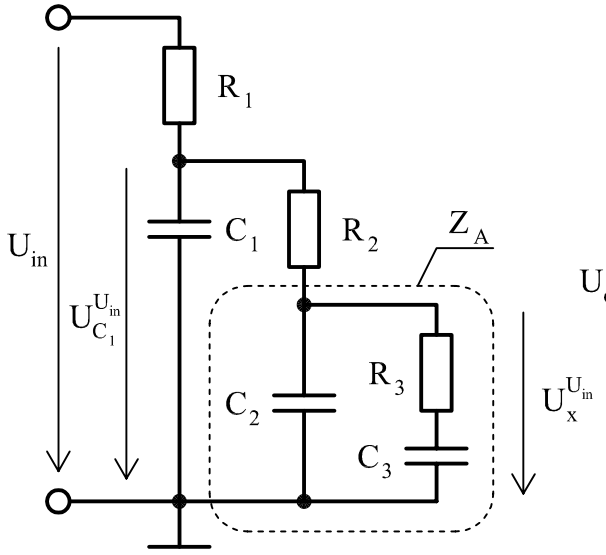
Při výpočtu $U_x^{U_{in}}$ položíme $U_{out} = 0$. Pak lze síť pasivních součástek překreslit dle obrázku 3.5. Stanovme nejprve výslednou impedanci kombinace C_2, R_3 a C_3:

$$Z_A = \frac{1}{pC_2} \Bigg\| \left(R_3 + \frac{1}{pC_3} \right) \tag{3.6}$$

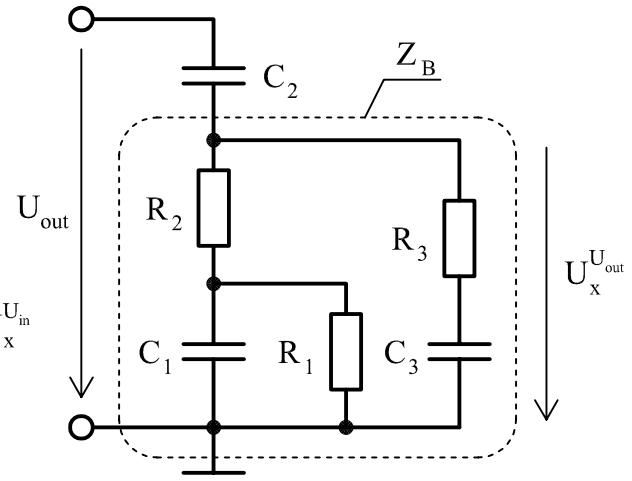
Pak

$$U_{C_1}^{U_{in}} = U_{in} \frac{\frac{1}{pC_1} \| (R_2 + Z_A)}{R_1 + \left[\frac{1}{pC_1} \| (R_2 + Z_A) \right]} \tag{3.7}$$

$$U_x^{U_{in}} = U_{C_1}^{U_{in}} \frac{Z_A}{R_2 + Z_A} \tag{3.8}$$



Obrázek 3.5: Stav obvodu pro $U_{out} = 0$



Obrázek 3.6: Stav obvodu pro $U_{in} = 0$

Obdobně při výpočtu $U_x^{U_{out}}$ položíme $U_{in} = 0$, síť pasivních součástek překreslíme dle obrázku 3.6 a stanovíme výslednou impedanci celé spodní části děliče:

$$Z_B = \left[R_2 + \left(\frac{1}{pC_1} \Bigg\| R_1 \right) \right] \Bigg\| \left(R_3 + \frac{1}{pC_3} \right) \tag{3.9}$$

Pak

$$U_x^{U_{out}} = U_{out} \frac{Z_B}{\frac{1}{pC_2} + Z_B} \tag{3.10}$$

Dle (3.5) pak určíme U_x a z něj napětí U_{C_3}:

$$U_{C_3} = U_x \frac{\frac{1}{pC_3}}{R_3 + \frac{1}{pC_3}} \tag{3.11}$$

Toto napětí vstupuje do operačního zesilovače, který je zapojen jako sledovač, a tedy

$$U_{out} = U_{C_3} \tag{3.12}$$

Přenos filtru je definován jako

$$F(p) = \frac{U_{out}}{U_{in}} \tag{3.13}$$

Ruční odvození přenosu filtru by bylo značně pracné, proto bylo provedeno automatické odvození pomocí skriptu `anafiltr.m`, do kterého byly zadány rovnice (3.5) až (3.13). Obdržíme výsledek ve tvaru

$$F(p) = \frac{1}{ap^3 + bp^2 + cp + 1} \tag{3.14}$$

kde

$$a = R_1 R_2 R_3 C_1 C_2 C_3 \tag{3.15}$$

$$b = R_1 R_2 C_1 C_3 + R_1 R_3 C_2 C_3 + R_1 R_3 C_1 C_3 + R_2 R_3 C_2 C_3 \tag{3.16}$$

$$c = R_1 C_1 + R_1 C_3 + R_2 C_3 + R_3 C_3 \tag{3.17}$$

Skript zároveň navrhne pro filtr 3. řádu s lomovým kmitočtem 4 kHz a s aproximací podle Butterwortha tyto hodnoty:

$$\begin{aligned} a &= 6,299.10^{-14} \\ b &= 3,166.10^{-9} \\ c &= 7,958.10^{-5} \end{aligned}$$

Nyní musíme nalézt takovou kombinaci hodnot součástek, pro kterou bude a, b, c nejblíže požadovaným hodnotám. Protože smysluplných kombinací není mnoho, byl výběr proveden ručně. Jako nejvhodnější se jeví kombinace

$$\begin{aligned} R_1 &= 12\ \text{k}\Omega & R_2 &= 10\ \text{k}\Omega & R_3 &= 10\ \text{k}\Omega \\ C_1 &= 4,7\ \text{nF} & C_2 &= 15\ \text{nF} & C_3 &= 680\ \text{pF} \end{aligned}$$

pro kterou vychází hodnoty

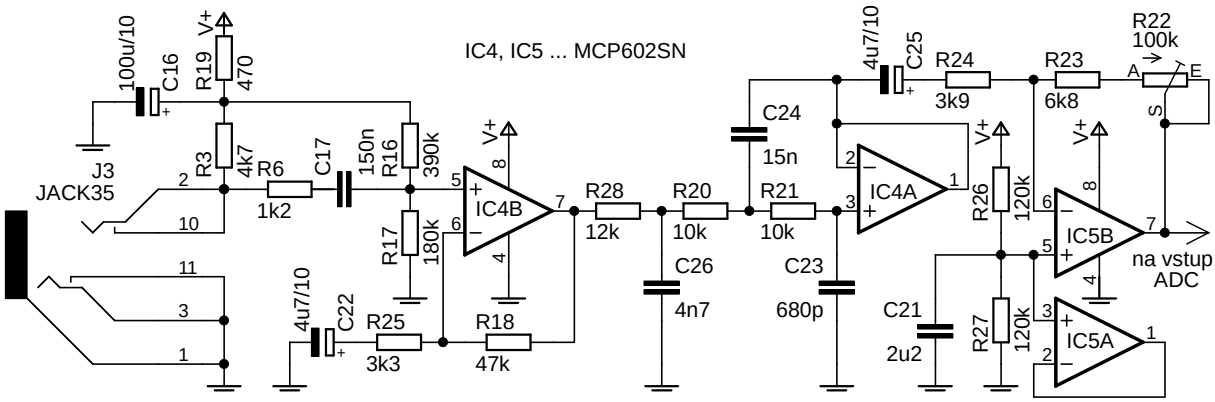
$$\begin{aligned} a &= 5,753.10^{-14} \\ b &= 3,011.10^{-9} \\ c &= 7,816.10^{-5} \end{aligned}$$

Amplitudová charakteristika filtru s těmito hodnotami součástek se od požadované charakteristiky vzdaluje nejvíce o 0,8 dB, jak je ukázáno ve skriptu `anafiltr.m`.

3.3.3. Celkové obvodové řešení

Vstupní analogový obvod musí kromě filtrace zajistit i zesílení slabého signálu z mikrofonu, abychom vhodně využili celý rozsah AD převodníku. Vstupní napětí převodníku musí ležet mezi 0 V a napájecím napětím mikrokontroléru (tj. 3,3 V). Experimentálně bylo zjištěno, že výstupní napětí elektretového mikrofonu dosahuje úrovně 8 až 120 mV (špička-špička) v závislosti na tom, z jaké vzdálenosti a jak hlasitě do mikrofonu mluvíme. Pro optimální využití převodníku tedy musíme signál z mikrofonu zesílit 28krát až 410krát v závislosti na situaci. Je tedy vhodné, aby zesílení bylo plynule nastavitelné.

Na základě tohoto rozboru byl navržen vstupní obvod dle obrázku 3.7 (jde o výřez z celkového schématu; označení součástek je proto jiné, než používá odvození v předcházející kapitole). Protože požadovaný rozkmit signálu na vstupu AD převodníku je



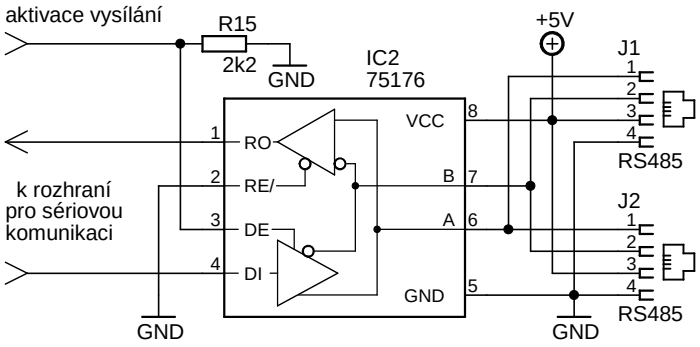
Obrázek 3.7: Vstupní obvod hlasového modulu

stejný jako napájecí napětí, jsou zde použity *rail-to-rail* operační zesilovače, které jsou schopny tento požadavek splnit. Elektretový mikrofon, připojený ke konektoru J3, je napájen přes rezistor R3. Článek R19+C16 zajišťuje přídavné vyhlazení napájecího napětí. Zesilovač IC4B je zapojen jako neinvertující a spolu s okolními součástkami zajišťuje zesílení vstupního signálu 15krát. Dělič R16+R17 nastavuje stejnosměrnou složku výstupního napětí zesilovače IC4B na asi 1 V. Následuje filtr dle návrhu v předchozí kapitole, který je postaven okolo IC4A. Za filtrem se nachází invertující zesilovač s IC5B, jehož zesílení lze nastavit pomocí trimru R22 v rozsahu 1,7krát až 27krát. Výstup tohoto zesilovače je přiveden na vstup AD převodníku v mikrokontroléru. Dělič R26+R27 zajišťuje, že stejnosměrná složka výstupního signálu je polovinou napájecího napětí. Zesilovač IC5A není využit.

3.4. Připojení ke sběrnici RS-485

Asynchronní sériová sběrnice RS-485 používá k přenosu dat dva vodiče (obvykle označované „A“ a „B“). O logické úrovni na sběrnici rozhoduje znaménko napětí mezi těmito vodiči (tj. to, který z vodičů je kladnější), nikoliv napětí vodičů vůči zemi. Díky tomu dosahuje sběrnice RS-485 velké odolnosti proti rušení, neboť pokud vedeme vodiče blízko u sebe (nejlépe jako kroucený pár), indukuje se do obou vodičů téměř stejné rušivé napětí a napětí mezi vodiči zůstane nezměněno. Proto se tato sběrnice často používá v průmyslové automatizaci. Pomocí sběrnice lze propojit několik zařízení, která mezi sebou komunikují v poloduplexním režimu. Komunikační protokol musí být navržen tak, aby vždy vysílalo pouze jedno zařízení a ostatní přijímala.

Připojení hlasového modulu ke sběrnici RS-485 je provedeno dle obrázku 3.8 pomocí diferenciální přijímače/budiče IC2. Tento integrovaný obvod je připojen k rozhraní pro sériovou komunikaci v mikrokontroléru. Budič sběrnice je zapínán a vypínán pomocí vstupu DE, který je taktéž připojen na jeden z pinů mikrokontroléru. Přijímač je trvale aktivní. Pull-down R15 zajišťuje deaktivaci budiče v čase inicializace nebo programování mikrokontroléru.



Obrázek 3.8: Obvod pro připojení ke sběrnice RS-485

Sběrnice RS-485 je vedena stejným kabelem jako napájení a připojuje se ke konektorům J1 a J2. Použití dvou konektorů, jejichž odpovídající piny jsou spojeny, umožňuje snadné propojení několika modulů pomocí přímých kabelů bez odboček. Budič IC2 je napájen přímo napětím 5 V (ostatní části modulu napájí stabilizátor 3,3 V), aby bylo dosaženo většího rozkmitu signálu na sběrnici. Napěťové úrovně mezi budičem a mikrokontrolérem není nutno převádět, neboť všechny vstupy mikrokontroléru jsou tolerantní k napětí 5 V.

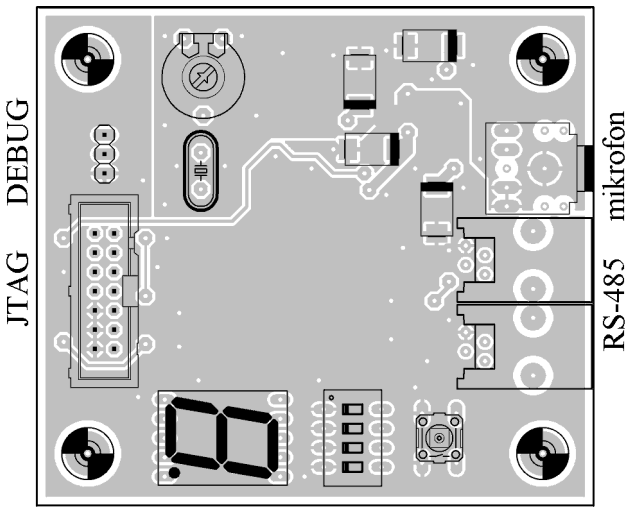
3.5. Deska plošných spojů

Při návrhu desky plošných spojů je třeba vycházet z technologie výroby desky. Vzhledem k tomu, že potřebujeme pouze jediný prototyp desky hlasového modulu, je deska vyrobena technologií nažehlování toneru z laserové tiskárny na základní materiál. Prokovy jsou nahrazeny zapájením drátu do předvrtaného otvoru. Touto technologií lze vyrobit nejvýše dvouvrstvou desku. Protože mikrokontrolér pracuje na kmitočtu 80 MHz, je třeba věnovat značnou pozornost rozvodu napájecího napětí. Proto byla zvolena následují koncepce desky:

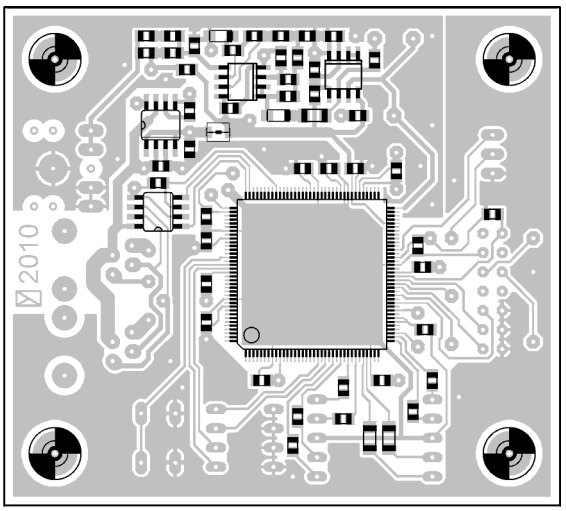
1. Vrchní strana desky slouží jako zemnící plocha. Zemnící plocha je provedena odděleně pro analogovou část (vstupní obvod) a pro digitální část (zbytek). Na této straně desky jsou umístěny všechny vývodové součástky a větší kondenzátory, které by vespod zbytečně zabíraly místo.

2. Spodní strana desky je osazena součástkami pro povrchovou montáž a zajišťuje propojení všech součástek. Nevyužitá místa jsou vyplněna zemnící plochou. Napětí 3,3 V je rozváděno napájecí plochou pod mikrokontrolérem.

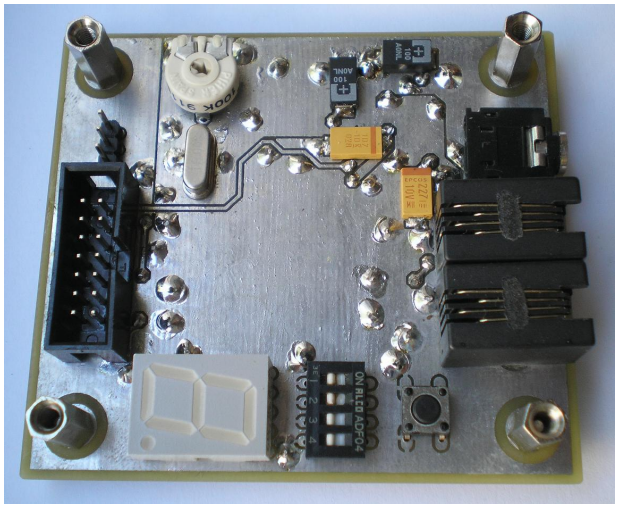
Návrh desky byl proveden v programu Eagle, výsledné soubory jsou součástí přílohy. V programu Eagle jsou strany desky zaměněny (tj. skutečná vrchní plocha desky je v hladině BOTTOM a naopak skutečná spodní plocha je v hladině TOP), aby bylo na složitější spodní stranu dobře vidět. Představu o výsledných obrazcích plošných spojů a rozmístění součástek dávají obrázky 3.9 a 3.10. Do obrázku 3.9 je zaneseno i rozmístění konektorů. Celkové rozměry desky jsou přibližně 73×66 mm, rozteč upevňovacích děr je 60×52 mm. Na obrázcích 3.11 a 3.12 je zobrazen skutečný vzhled hlasového modulu.



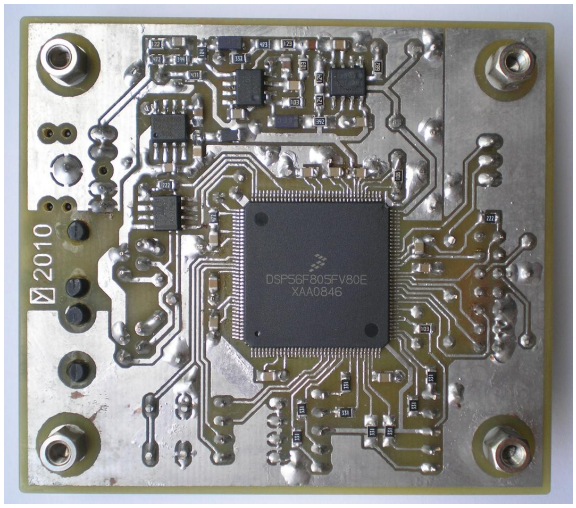
Obrázek 3.9: Rozložení součástek na vrchní straně hlasového modulu



Obrázek 3.10: Rozložení součástek na spodní straně hlasového modulu



Obrázek 3.11: Fotografie modulu shora



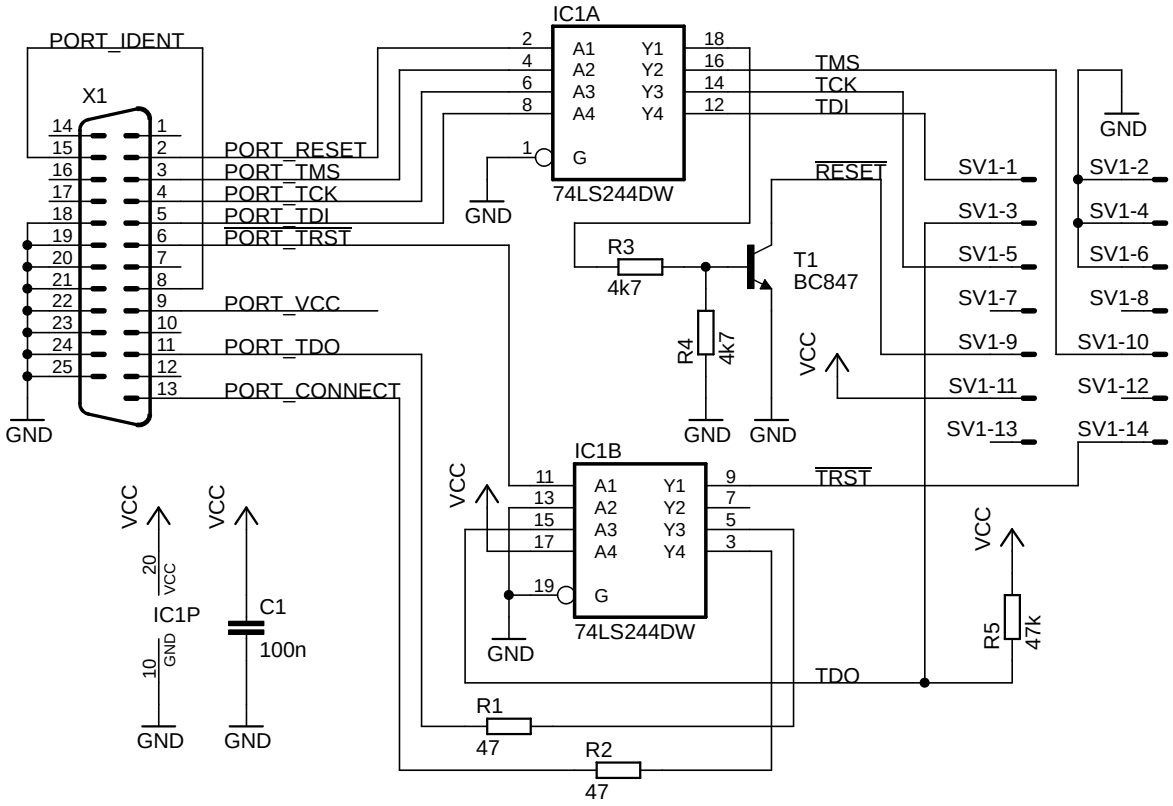
Obrázek 3.12: Fotografie modulu zespodu

3.6. Programátor

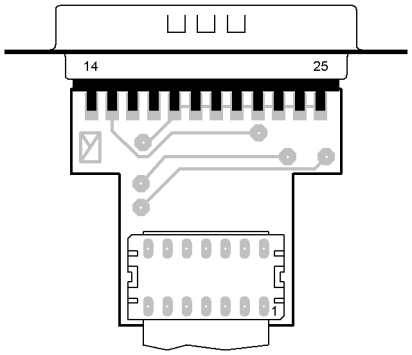
Pro přenos hotového programu do mikrokontroléru je zapotřebí programátor. Pokud nechceme kupovat profesionální výrobek, lze mikrokontrolér naprogramovat způsobem, který firma Freescale používá na svých vývojových deskách, např. [7]. Princip je ten, že osobní počítač na paralelním portu přímo generuje signály programovacího rozhraní JTAG. Teoreticky tedy stačí propojit odpovídající piny paralelního portu a mikrokontroléru, nejsou zapotřebí žádné přídavné logické obvody. V praxi je ovšem vhodné oddělit obě zařízení alespoň jednoduchým TTL budičem, aby při chybě na straně programovaného hardware (např. přepólování napájecího napětí) nebyl ohrožen osobní počítač.

Na základě výše uvedených informací byl navržen programátor dle obrázku 3.13. Kvůli jednoduchosti je programátor napájen z programovaného zařízení. Programátor byl realizován na oboustranné desce plošných spojů dle obrázků 3.14 a 3.15. Rozměry desky

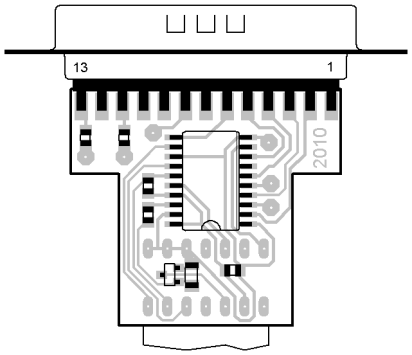
byly zvoleny tak, že celou desku lze umístit do krytu konektoru DSUB-25 (paralelní port), samotný konektor je připájen na nejdelší straně desky. Z opačné strany je vyveden plochý kabel, na němž je umístěn samořezný konektor. Ten zajišťuje připojení k desce hlasového modulu.



Obrázek 3.13: Schéma zapojení programátoru



Obrázek 3.14: Rozložení součástek na vrchní straně desky programátoru



Obrázek 3.15: Rozložení součástek na spodní straně desky programátoru

4. Návrh software

Jak bylo uvedeno v kapitole 3.1, kontroléry řady 56800 obsahují poměrně výkonné jádro, umožňující efektivní provádění operací typických pro zpracování signálu. Toto jádro je určeno pro práci s čísly v reprezentaci s pevnou řádovou čárkou (anglicky *fixed point*). Výpočty v plovoucí čárce (*floating point*) jsou na tomto jádru samozřejmě také možné, musí však být emulovány softwarově a proto jsou řádově pomalejší, což pro danou aplikaci není únosné. Proto budou nejprve popsány číselné reprezentace, které umí jádro efektivně využívat. Dále bude popsán způsob práce jádra. Teprve poté bude přistoupeno k návrhu vlastního software.

4.1. Výpočetní jádro kontrolérů řady 56800

4.1.1. Reprezentace čísel

Obvody řady 56800 jsou 16-bitové signálové kontroléry, proto při výpočtech primárně pracují se slovy o šířce 16 bitů. Při reprezentaci čísla s pevnou řádovou čárkou má každý bit $b[i]$ v 16-bitovém slovu pevně přiřazenou váhu $w[i]$, kde $i \in \{0, 1, \ldots, 15\}$ je pozice bitu. Číselnou hodnotu x, kterou reprezentuje 16-bitové slovo, zjistíme užitím vztahu

$$x = \sum_{i=0}^{15} w[i].b[i] \tag{4.1}$$

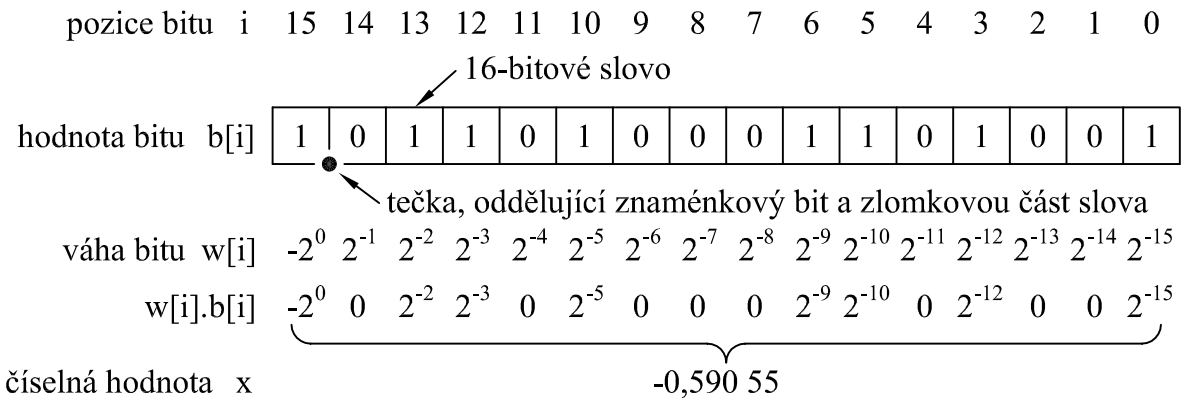
Pokud zvolíme váhy $w[i]$ takto

$$w[i] = \begin{cases} 2^{i-15} & \text{pro } i \in \{0, 1, \ldots, 14\} \\ -2^0 & \text{pro } i = 15 \end{cases} \tag{4.2}$$

pak může 16-bitové slovo reprezentovat čísla od -1 do $(1 - 2^{-15}) \doteq 0{,}99997$ s krokem (tj. vzdáleností sousedních hodnot) $2^{-15} \doteq 0{,}000\,03$. Jakékoliv veličiny, se kterým chceme pracovat, musíme tedy normalizovat do intervalu $\langle -1; 1)$ (hodnotu přesně 1 není možno pomocí dané reprezentace vyjádřit, což ovšem není většinou na závadu). Povšimněme si, že bit $b[15]$ rozhoduje, zda slovo reprezentuje nezáporné číslo (pokud $b[15] = 0$) nebo záporné číslo (pokud $b[15] = 1$). Nejvyšší bit $b[15]$ tedy plní v tomto případě funkci *znaménkového bitu*. Výše uvedené principy jsou demonstrovány příkladem na obrázku 4.1. Tímto způsobem je reprezentována většina vstupních a výstupních hodnot ve výpočtech.

V některých případech může být přesnost 16-bitové reprezentace nedostatečná. Například násobíme-li dvě 16-bitová čísla, z nichž každé obsahuje 15 bitů vpravo od tečky, pak obdržíme výsledek, který má 30 bitů vpravo od tečky. Pokud nechceme ztrácet přesnost zaokrouhlováním, použijeme pro reprezentaci hodnoty 32-bitové dvojslovo, ve kterém jsou váhy jednotlivých bitů definovány takto

$$w[i] = \begin{cases} 2^{i-31} & \text{pro } i \in \{0, 1, \ldots, 30\} \\ -2^0 & \text{pro } i = 31 \end{cases} \tag{4.3}$$



Obrázek 4.1: Příklad reprezentace číselné hodnoty pomocí 16-bitového slova

Tímto způsobem můžeme pomocí 32-bitového dvojslova reprezentovat čísla od -1 do $(1 - 2^{-31})$ s krokem (tj. vzdáleností sousedních hodnot) $2^{-31} \doteq 4{,}7.10^{-10}$, tedy výrazně přesněji, než s použitím 16-bitového slova.

Díky tomu, že při výpočtech pracujeme s čísly v intervalu $\langle -1; 1)$, nemůže při násobení dvou čísel nikdy dojít k přetečení – součin bude v absolutní hodnotě vždy menší nebo stejný jako násobené hodnoty. Při sčítání ovšem k přetečení dojít může. Proto se v registrech jádra, zvaných *akumulátory*, v některých případech používá 36-bitová reprezentace čísel, u které jsou váhy jednotlivých bitů definovány

$$w[i] = \begin{cases} 2^{i-31} & \text{pro } i \in \{0, 1, \ldots, 34\} \\ -2^4 & \text{pro } i = 35 \end{cases} \tag{4.4}$$

Takto můžeme v 36-bitovém akumulátoru reprezentovat čísla od -16 do $(16 - 2^{-31})$ s krokem (tj. vzdáleností sousedních hodnot) $2^{-31} \doteq 4{,}7.10^{-10}$. Z toho plyne, že v akumulátoru můžeme uložit součet až šestnácti libovolných 32-bitových dvojslov, přičemž je zajištěno, že nedojde k přetečení výsledku.

4.1.2. Struktura jádra

Výpočetní jádro kontrolérů řady 56800 se skládá z několika jednotek. Pro další popis je podstatná pouze *aritmeticko-logická jednotka*, *jednotka generování adres* a *jednotka pro hardwarové smyčky*. Podrobnější popis lze nalézt v [8].

Aritmeticko-logická jednotka zajišťuje provádění veškerých výpočtů se zpracovávanými daty. Pro uložení vstupních dat slouží tři 16-bitové registry, označené X0, Y0 a Y1. Pro uložení mezivýsledků a výstupních dat slouží dva 36-bitové akumulátory A a B. Klíčovou součástí je jednotka MAC (zkratka od *multiply-accumulate*), která umí v jediném instrukčním cyklu vynásobit dva šestnáctibitové registry a výsledek násobení přičíst k obsahu akumulátoru. Volitelně je možno zapnout saturaci výsledku výpočtu. To znamená, že pokud dojde během výpočtu k přetečení, je nesprávný výsledek nahrazen maximální nebo minimální reprezentovatelnou hodnotou (podle směru přetečení), což obvykle vede na menší celkovou chybu výpočtu než neošetřené přetečení.

Jednotka generování adres řídí přístup do datové paměti. Jejím úkolem je zásobovat aritmeticko-logickou jednotku vstupními daty nebo naopak vypočtené hodnoty ukládat do datové paměti. Díky tomu, že jádro je spojeno s datovou pamětí pomocí dvou adresových a dvou datových sběrnic, mohou být během jednoho instrukčního cyklu načteny až dvě 16-bitové hodnoty. Jednotka generování adres obsahuje mimo jiné i registry R0, R1, R2 a R3, které určují adresy v datové paměti, se kterými právě pracujeme. Obsah těchto registrů je možno v každém cyklu inkrementovat či dekrementovat a tím postupně zpracovat souvislou oblast paměti. Dále tato jednotka umožňuje použít tzv. *modulo adresování*, které lze použít k efektivní implementaci kruhového bufferu, jak bude ukázáno v kapitole 4.3.2.

Jednotka pro hardwarové smyčky slouží pro efektivní realizaci programových smyček se známým počtem průchodů (v běžných programovacích jazycích se jedná o smyčky *for*). Požadavek na opakování určité části kódu vyžaduje u běžného procesoru vytvoření pomocné proměnné. Tuto proměnnou na začátku inicializujeme na požadovaný počet opakování. Při každém průchodu smyčkou tuto proměnnou dekrementujeme a při nenulové hodnotě proměnné skočíme na začátek cyklu. Jakmile proměnná dosáhne nuly, cyklus ukončíme. Neustálé dekrementování a testování proměnné u běžného procesoru zpomaluje program. U kontrolérů řady 56800 ovšem jednotka pro hardwarové smyčky umí výše popsané operace provádět nazávisle, bez účasti ostatních částí jádra. Tím lze provádění smyček výrazně urychlit.

Vysokého výpočetního výkonu kontrolérů řady 56800 je dosaženo zejména součinností a paralelním chodem výše uvedených jednotek. Příkladem může být instrukce

```
MAC Y0,X0,A   X:(R0)+,Y0   X:(R3)+,X0
```

která v jediném instrukčním cyklu provede toto:

1. Z registrů Y0 a X0 jsou načtena dvě 16-bitová slova a vynásobena mezi sebou.
2. 32-bitový výsledek násobení je přičten k 36-bitovému obsahu akumulátoru A.
3. Z adresy v datové paměti, na kterou odkazuje registr R0, je načteno 16-bitové slovo, které je uloženo do registru Y0. Tato hodnota může být využita v dalším cyklu.
4. Adresa v registru R0 je zvětšena o jedničku.
5. Z adresy v datové paměti, na kterou odkazuje registr R3, je načteno 16-bitové slovo, které je uloženo do registru X0. Tato hodnota může být využita v dalším cyklu.
6. Adresa v registru R3 je zvětšena o jedničku.

Instrukci je možno samozřejmě několikrát opakovat s využitím hardwarové smyčky, přičemž počet opakování je prakticky libovolný. Tohoto postupu lze použít k výpočtu skalárního součinu dvou vektorů, který je v různých obměnách často používán při zpracování

signálu (vážený aritmetický průměr, korelace, konvoluce, energie v signálu, odezva FIR filtru atd.). Pro délku vstupních vektorů n trvá celý výpočet pouze n instrukčních cyklů (nepočítáme-li inicializaci výpočtu). Přitom jsou využity všechny tři výše popsané jednotky v jádře, přičemž

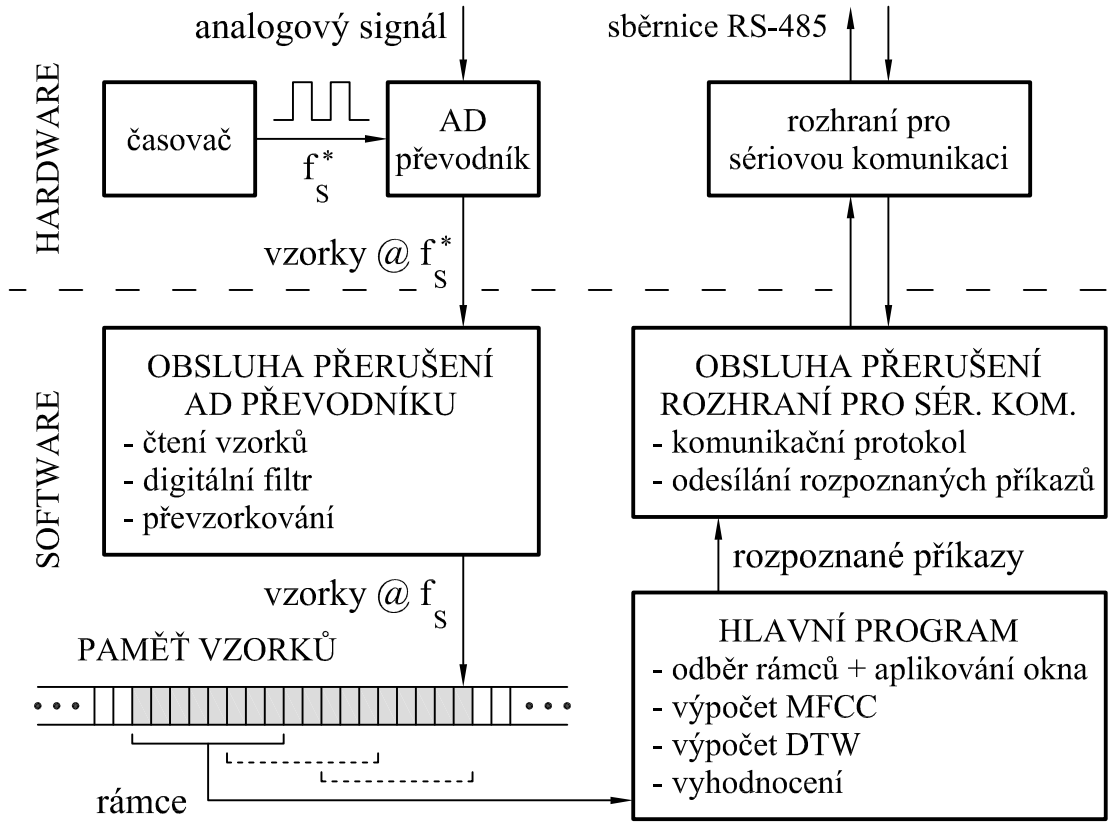
- jednotka generování adres zajišťuje postupné čtení obou vektorů prvek po prvku a předávání těchto prvků do aritmeticko-logické jednotky
- aritmeticko-logická jednotka násobí odpovídající prvky vektorů a součin přičítá k celkovému výsledku
- jednotka pro hardwarové smyčky zajistí ukončení výpočtu, jakmile zpracujeme poslední prvky obou vektorů

Samozřejmě předpokládáme, že prvky každého vstupního vektoru jsou uloženy v datové paměti postupně za sebou.

4.2. Rámcové rozvržení software

Obrázek 4.2 znázorňuje rozdělení všech požadovaných úloh mezi periferie mikrokontroléru a jednotlivé části software v režimu rozpoznávání (další režimy budou popsány v kapitole 4.6). Začátek naznačeného řetězce vpodstatě realizuje digitální část obrázku 3.2. Analogový signál je převáděn na digitální pomocí AD převodníku. Činnost AD převodníku je řízena pomocí jednoho z časovačů, který generuje signál o frekveci $f_S^* = 32$ kHz pro spouštění převodníku. Výstupem převodníku je sekvence vzorků. Při každém dokončení AD převodu je volána obsluha přerušení AD převodníku, která načte aktuální vzorek a zpracuje jej pomocí digitálního filtru. Výstup filtru je zároveň převzorkován na frekvenci $f_S = 8$ kHz. To znamená, že při každém čtvrtém volání obsluhy přerušení AD převodníku získáme jeden výstupní vzorek.

Výstupní vzorky, generované s frekvencí $f_S = 8$ kHz, jsou ukládány na konec paměti vzorků. Ze začátku paměti vzorků jsou pak vzorky odebírány hlavním programem. Způsob odebírání vzorků koresponduje s tvorbou rámců. Vzorky jsou odebrány tehdy, jakmile je jich k dispozici dostatečný počet pro vytvoření jednoho rámce a jakmile není hlavní program zaneprázdněn zpracováním předchozího rámce. Hlavní program provádí nejvýznamnější část zpracování, tj. zajišťuje aplikování okna na rámec, výpočet MFCC, porovnání se vzory pomocí DTW a vyhodnocení. Výstupem hlavního programu jsou rozpoznané hlasové příkazy. Komunikace s nadřazeným systémem probíhá pomocí sběrnice RS-485 a rozhraní pro sériovou komunikaci. Příjem nebo vyslání znaku tímto rozhraním spouští příslušnou obsluhu přerušení, která zajišťuje řízení komunikace dle použitého protokolu. V případě, že si nadřazený systém vyžádá informaci o rozpoznaných příkazech, převezme obsluha přerušení tuto informaci od hlavního programu a pošle ji nadřazenému systému.



Obrázek 4.2: Blokové schéma software s návazností na hardware

Pro vývoj software hlasového modulu bylo použito prostředí *Freescale CodeWarrior* s nadstavbou *Processor Expert*. Prostředí *CodeWarrior* umožňuje vývoj aplikací jak v asembleru, tak v jazyce C. Tyto způsoby zápisu byly dle potřeby kombinovány. Větší část zpracování dat je programována v asembleru, který umožňuje efektivnější zápis operací v pevné čárce. Zápis v C byl použit pro definování proměnných a pro ta místa programu, kde upřednostňujeme přehlednost před rychlostí provádění (např. stavový automat v kapitole 4.6). Veškerý zdrojový kód je součástí přílohy.

Processor Expert je nástroj, sloužící k automatickému generování inicializačního kódu, ve kterém stačí nakonfigurovat vybranou periferii pomocí grafického rozhraní a není nutno se zabývat ručním nastavováním příslušných registrů. *Processor Expert* dále umožňuje snadnější obsluhu periferií prostřednictvím předpřipravených ovladačů; tento přístup byl ale v některých případech shledán jako poměrně těžkopádný a byl proto využit jen částečně.

4.3. Obsluha přerušení AD převodníku

Připomeňme, že úkolem obsluhy přerušení AD převodníku je načíst vzorek z AD převodníku, zpracovat jej pomocí digitálního filtru a výstupní vzorek z digitálního filtru uložit do paměti vzorků. Samotné čtení vzorků z AD převodníku je triviální operací, proto se dále budeme věnovat výhradně digitálnímu filtru a zápisu do paměti vzorků.

4.3.1. Digitální filtr

Úkolem digitálního filtru typu dolní propust je doplňovat analogový antialiasingový filtr, jak bylo uvedeno na obrázku 3.3.

Na obrázku 3.2 je digitální filtr pro jednoduchost naznačen jako samostatný blok, nezávislý na následném převzorkování. Pokud chceme dosáhnout velké strmosti přenosové charakteristiky filtru i při použití nízkého řádu filtru, je vhodnější tyto dva kroky výpočtu vzájemně propojit. Konkrétně: pokud chceme vzorkovací frekvenci celkově redukovat na čtvrtinu, je vhodné zpracovat signál nejprve prvním stupněm filtru, pak redukovat vzorkovací frekvenci na polovinu, zpracovat signál druhým stupněm filtru a opětovně redukovat vzorkovací frekvenci na polovinu. Řád filtru můžeme dále minimalizovat tak, že filtr navrhneme jako vícefázový, tj. sudé a liché vzorky zpracujeme v každém stupni zvlášť samostatným filtrem, jejichž výstupy pak sečteme.

IIR filtr dle výše uvedených principů byl navržen pomocí skriptu `digfiltr.m`. Návrh filtru vychází z [9]. Byly zvoleny následující specifikace filtru:

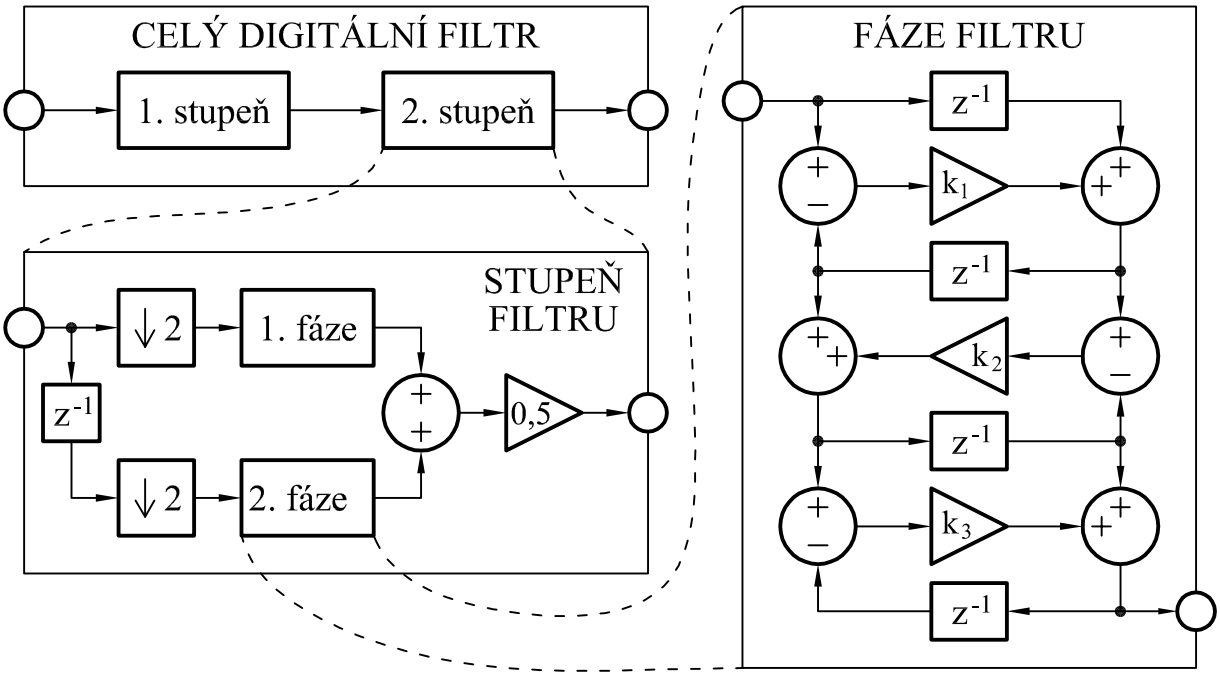
- filtr propustí všechny frekvence pod 3 840 Hz
- filtr potlačí všechny frekvence nad 4 160 Hz
- potlačení frekvencí v nepropustné části charakteristiky bude nejméně 60 dB
- výstupní vzorkovací frekvence je čtvrtinou vstupní vzorkovací frekvence

I pro tyto poměrně přísné specifikace má filtr pouze 12 stavů a 8 koeficientů. Získání jednoho výstupního vzorku vyžaduje 10 operací násobení a 20 operací sčítání.

Celková struktura digitálního filtru je na obrázku 4.3. Filtr se skládá ze dvou stupňů. V každém stupni se nejprve přicházející vzorky rozdělí na sudé a liché. Sudé a liché vzorky jsou nezávisle zpracovány první a druhou fází příslušného stupně. Součet (respektive průměr) výstupů jednotlivých fází tvoří výstup stupně filtru. Každá fáze filtru je tvořena „žebříkovou“ strukturou, znázorněnou na obrázku 4.3 vpravo. Každá fáze druhého stupně je tvořena čtyřmi stavy, třemi koeficienty (zesíleními), třemi sčítačkami a třemi odečítačkami. Koeficienty v první a druhé fázi jsou odlišné. Fáze prvního stupně vypadají obdobně, ale obsahují pouze dva stavy, jeden koeficient, jednu sčítačku a jednu odečítačku.

Použitá metoda návrhu filtru předpokládá, že všechny výpočty ve filtru probíhají na množině reálných čísel, tj. s nekonečnou přesností a bez omezení rozsahu číselných hodnot. Ve skutečnosti jsou ale všechny koeficienty i stavy filtru reprezentovány s užitím 16-bitových slov. Je tedy nutno ověřit, zda tento fakt nezmění přenosovou charakteristiku filtru. Testování filtru bylo provedeno v prostředí Simulink (soubor `kvantdigfiltr.mdl`). Filtr pracuje bezchybně, pokud vstupní hodnoty leží v intervalu $\langle -0{,}5; 0{,}5)$. Není tedy možno využít celý rozsah 16-bitového slova, který činí $\langle -1; 1)$. Toto ovšem není na závadu, neboť AD převodník produkuje hodnoty v intervalu $\langle 0; 1)$. Pomocí offsetového registru přímo v AD převodníku tento interval snadno posuneme do $\langle -0{,}5; 0{,}5)$, což je právě povolený rozsah vstupu digitálního filtru.

Výpočet digitálního filtru probíhá v rámci obsluhy přerušení. U každé obsluhy přerušení musíme na začátku uložit všechny používané registry do zásobníku a na konci obsluhy



Obrázek 4.3: Struktura digitálního filtru

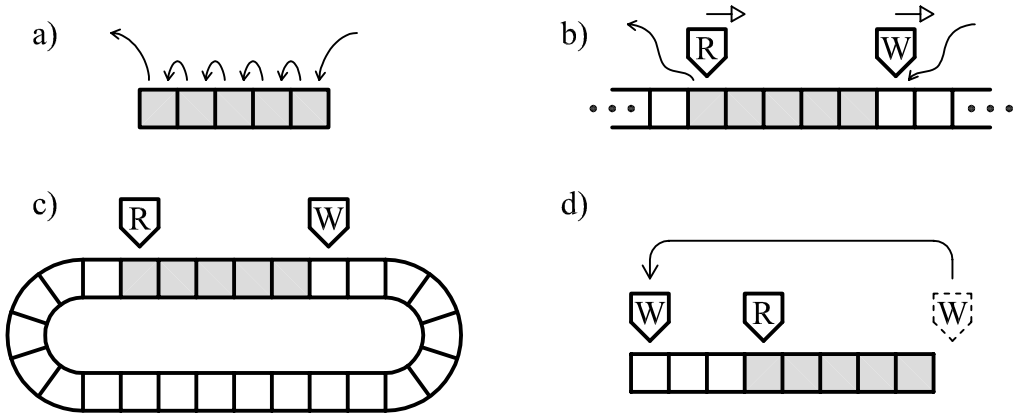
přerušení provést jejich obnovení. Pokud bychom toto neprovedli, došlo by k nežádoucímu ovlivnění chodu hlavního programu. Problém je v tom, že výpočet digitálního filtru vyžaduje zálohování všech registrů aritmeticko-logické jednotky. Uložení a obnovení těchto registrů by trvalo poměrně velký počet instrukčních cyklů, který je řádově srovnatelný s výpočtem digitálního filtru. To by znamenalo významné zpomalení programu.

Řešení spočívá v tom, že vzorky z AD převodníku ukládáme do paměti, namísto toho, abychom je ihned zpracovali. Uložení do paměti nevyžaduje plnou zálohu registrů aritmeticko-logické jednotky. Teprve tehdy, když takto shromáždíme čtyři vstupní vzorky (což nastane každé čtvrté volání obsluhy přerušení), tak zálohujeme registry, provedeme výpočet filtru pro celou skupinu čtyř vzorků, získáme jeden výstupní vzorek a obnovíme stav registrů. Díky tomu redukujeme čas nutný k zálohování registrů na čtvrtinu původní hodnoty. Vedlejším efektem je zpřehlednění výpočtu digitálního filtru, neboť se nemusíme rozhodovat, které fáze a stupně filtru se mají při konkrétním volání obsluhy přerušení počítat.

4.3.2. Paměť vzorků

Paměť vzorků musí být v principu typu FIFO, tj. vzorek, který byl do paměti uložen jako první, je z paměti jako první i vyzvednut. Toto lze realizovat například dle obrázku 4.4 a), kde vždy před zápisem dat do paměťové buňky zcela vpravo posuneme obsah všech buněk o jedno místo doleva a z buňky vlevo odebereme její obsah. Tento princip však znamená velké množství přístupů do paměti, navíc neumožňuje provádět zápis a čtení vzorku nezávisle (v paměti je vždy konstantní množství dat). Proto tento základní způsob nebyl použit.

Namísto toho, abychom pohybovali obsahem paměťových buněk, můžeme pohybovat místem, do kterého zapisujeme data a místem, ze kterého čteme data (tato místa obvykle reprezentujeme pomocí ukazatelů). Tento princip je ukázán na obrázku 4.4 b). Data ukládáme do buňky, na kterou ukazuje ukazatel pro zápis (označen W) a poté ukazatel pro zápis posuneme. Obdobně při čtení vyzvedneme data z místa, kam ukazuje ukazatel pro čtení (označen R), a pak ukazatel posuneme.



Obrázek 4.4: Odvození principu kruhového bufferu

Princip čistě dle obrázku 4.4 b) by pro trvalou činnost vyžadoval nekonečné množství paměti, neboť ukazatel pro zápis se neustále posouvá k vyšším adresám v paměti. Zároveň se ale neustále uvolňuje místo vlevo od ukazatele pro čtení. Proto můžeme vzájemně napojit paměťové buňky vpravo od ukazatele pro zápis a vlevo od ukazatele pro čtení, čímž vznikne kruhový buffer dle obrázku 4.4 c).

Paměť mikrokontroléru je ovšem adresována lineárně, tj. neobsahuje žádné smyčky dle obrázku 4.4 c). Proto musíme spojení paměťových buněk vytvořit uměle, jak je naznačeno na obrázku 4.4 d). Pokaždé, když se jeden z ukazatelů dostane za poslední buňku, tj. mimo paměť vyhrazenou pro buffer, přemístíme jej na začátek bufferu. U kontrolérů řady 56800 lze tuto operaci za určitých podmínek řešit hardwarově. Využíváme při tom tzv. *modulo adresování*, které bylo zmíněno v kapitole 4.1.2. Z pohledu programátora se pak buffer skutečně chová jako nekonečná smyčka dle obrázku 4.4 c).

Vzhledem k tomu, že z bufferu odebírá hlavní program rámce o délce 128 vzorků, musí být velikost bufferu větší než tato hodnota. Proto byla zvolena velikost kruhového bufferu 256 vzorků. Počet vzorků, uložených aktuálně v bufferu, je možno odvodit z polohy ukazatele pro zápis a pro čtení. Jako praktičtější se ovšem ukázalo uložení počtu vzorků do samostatné proměnné. Při každém zápisu nového vzorku je tato proměnná kontrolována a pokud hrozí přeplnění bufferu (tj. hlavní program nestíhá odebírat rámce), dojde ke smazání nejstaršího vzorku, čímž se uvolní místo pro zápis. K této situaci by však teoreticky nemělo nikdy dojít, jedná se spíše o „pojistku“ při ladění programu.

4.4. Hlavní program

4.4.1. Odběr rámců

Prvním úkolem hlavního programu je zajistit odběr rámců z paměti vzorků. Rámec je možno odebrat tehdy, pokud je v paměti minimálně 128 vzorků; hlavní program tedy obsahuje smyčku, ve které čeká na splnění této podmínky. Rámce se mají částečně překrývat (dle obrázku 2.5). Proto ukazatel pro čtení (zmiňovaný v předchozí kapitole) vždy posuneme o 80 vzorků, ale ve skutečnosti z paměti přečteme 128 vzorků. Díky tomu některé vzorky použijeme opakovaně, jak je požadováno.

Během kopírování vzorků zároveň na rámec aplikujeme okno, definované vztahem (2.6). Protože vyčíslení definiční funkce okna je relativně náročné, je vektor okna předpřipraven v paměti FLASH. Díky symetrii okna může být uložena pouze jeho polovina.

4.4.2. Fourierova transformace

Pro výpočet Fourierovy transformace rámce byla použita funkce `rfft` z knihovny [10], která je součástí vývojového prostředí *CodeWarrior*. Proto se v dalším textu omezíme jen na základní princip činnosti bez korektního matematického odvození.

Použitá funkce využívá algoritmus rychlé Fourierovy transformace (FFT), jehož nosná myšlenka je tato [11]: mějme řadu vzorků $x[n]$ kde $n \in \{0, 1, \ldots, N-1\}$. Předpokládejme $N = 2^i$, $i \in \{1, 2, 3, \ldots\}$. Chceme-li vypočíst rychlou Fourierovu transformaci této řady, rozdělíme tuto řadu vzorků na dvě řady $y[k]$ a $z[k]$, kde $k \in \{0, 1, \ldots, N/2-1\}$, přičemž $y[k]$ obsahuje sudé vzorky z $x[n]$ a $z[k]$ obsahuje liché vzorky z $x[n]$, tedy

$$y[k] = x[2k] \tag{4.5}$$

$$z[k] = x[2k+1] \tag{4.6}$$

Pokud $Y[k]$ a $Z[k]$ představují Fourierovy transformace řad $y[k]$ a $z[k]$, lze pak Fourierovu transformaci $X[n]$ původní řady $x[n]$ určit jako

$$X[k] = Y[k] + W^k Z[k] \tag{4.7}$$

$$X[k + N/2] = Y[k] - W^k Z[k] \tag{4.8}$$

kde

$$W = \mathrm{e}^{-2\pi \mathrm{j}/N} \tag{4.9}$$

Výpočet Fourierovy transformace dlouhé řady tedy nahradíme výpočtem Fourierových transformací dvou kratších řad. Tuto techniku lze ovšem použít opakovaně i na řady $y[k]$ a $z[k]$, takže pokud měla původní řada $x[n]$ celkem $N = 2^i$ vzorků, tak po i-tém opakování tohoto postupu musíme určit celkem 2^i Fourierových transformací „řad“ o jednom vzorku. „Výpočet“ Fourierovy transformace je však pro řadu o jednom vzorku triviální – je to přímo hodnota onoho jediného vzorku.

Hodnoty W^k jsou pro dané N vypočteny dopředu a uloženy ve vyhledávací tabulce. S použitím uvedeného postupu je pro výpočet Fourierovy transformace řady N vzorků

zapotřebí řádově $(N.\log_2 N)$ operací, což je pro velké N výrazně méně, než výpočet dle definičního vztahu (2.3), který vyžaduje řádově N^2 operací. Při vhodném uspořádání operací lze celý výpočet provést tzv. *in-place*. To znamená, že během výpočtu postupně přepisujeme vstupní vektor jednotlivými mezivýsledky a konečný výsledek obdržíme bez použití přídavného „odkládacího prostoru“. Popsaný algoritmus má tedy menší nároky na výpočetní výkon i na paměť, než základní vztah (2.3). Zde je nutno podotknout, že do funkce vstupuje vektor reálných čísel a výsledkem je vektor komplexních čísel, který by měl zabírat dvojnásobek paměti, neboť musíme zvlášť uložit reálnou a imaginární složku. Protože je ale výsledné spektrum symetrické, nemusí se polovina výsledného vektoru ukládat a množství dat na vstupu i na výstupu je stejné.

Analyzujme nyní vztah (4.7) a tím i celý algoritmus FFT z hlediska možnosti přetečení při reprezentaci čísel v pevné čárce. Obě složky komplexního čísla ukládáme jako 16-bitová slova a mohou tedy nabývat hodnot v intervalu $\langle -1; 1)$. Nejprve analyzujme násobení $(W^k).(Z[k])$, které vlastně představuje „otočení“ čísla v komplexní rovině. Absolutní hodnota čísla $Z[k]$ se při tom nemění, pouze dochází k „přelévání“ mezi reálnou a imaginární složkou čísla. Nejhorší možný případ z hlediska přetečení je tento:

$$(W^k).(Z[k]) = \left(\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}\mathrm{j}\right).(-1 - 1\mathrm{j}) = -\sqrt{2} \tag{4.10}$$

V tomto případě se reálná i imaginární složka při pootočení „slije“ pouze do reálné složky, což vede k přetečení, neboť výsledek $-\sqrt{2} \doteq -1,41$ nelze pomocí používané reprezentace vyjádřit. Je zřejmé, že aby k přetečení nemohlo dojít, stačí aby všechny vstupní prvky splňovaly podmínku $|Z[k]| \leq 1$.

Při výpočtu součtu $(Y[k]) + (W^k Z[k])$ je typově nejhorším případem

$$(Y[k]) + (W^k Z[k]) = (-1) + (-1) = -2 \tag{4.11}$$

Řešením by bylo po každém součtu provést vydělení výsledku dvěma. To by ale vedlo např. pro $N = 128$ k tomu, že bychom výsledek Fourierovy transformace obdrželi 128krát zmenšený, neboť dělení dvěma použijeme celkem 7krát. Při slabém vstupním signálu by tak některé frekvenční složky mohly zcela zaniknout nebo by jejich obsah byl zkreslen díky zaokrouhlovacím chybám. Použitá funkce `rfft` nabízí 3 varianty řešení tohoto problému:

1. Po sčítání se dělení dvěma nikdy neprovádí. Uživatel musí signál vstupující do funkce `rfft` upravit takovým způsobem, aby během výpočtu nemohlo dojít k přetečení.

2. Po sčítání se dělení dvěma vždy provádí. K přetečení tedy za žádných okolností nemůže dojít, pro slabý vstupní signál však může dojít ke ztrátě přesnosti výpočtu.

3. Po sčítání se dělení dvěma provede tehdy, pokud by při dalším výpočtu mohlo dojít k přetečení. Tato možnost tedy zabraňuje vzniku přetečení, ale přitom zbytečně nezmenšuje slabý signál. Rozhodnutí o dělení či nedělení je společné pro všechna data a provádí se vždy po jednom „průchodu“ FFT algoritmu. Funkce při návratu předává informaci o počtu uskutečněných dělení.

Pro software hlasového modulu byla zvolena třetí možnost. To, že jsou jednotlivé rámce jinak zesíleny či zeslabeny, kompenzujeme až po logaritmování, prakticky celý výpočet MFCC tak probíhá s optimálním využitím rozsahu reprezentace čísel.

4.4.3. Výpočet druhé mocniny absolutní hodnoty

V dalším kroku z výstupu Fourierovy transformace, což je vektor komplexních čísel, vytvoříme vektor druhých mocnin absolutních hodnot. Tato operace je triviální, neboť pro komplexní číslo Q s reálnou složkou Q_{Re} a s imaginární složkou Q_{Im} platí

$$|Q|^2 = \left(\sqrt{Q_{Re}^2 + Q_{Im}^2}\right)^2 = Q_{Re}^2 + Q_{Im}^2 = Q_{Re}.Q_{Re} + Q_{Im}.Q_{Im} \tag{4.12}$$

Výsledek je třeba uložit do 32-bitového dvojslova, neboť při násobení dochází ke zvětšování počtu platných bitů, jak je popsáno v kapitole 4.1.1. Při uložení do 16-bitového slova bychom pro $|Q| < 0{,}005$ vždy obdrželi nulový výsledek, čímž bychom zcela potlačili méně výrazné frekvenční složky. Protože do výpočtu vstupují dvě 16-bitová slova a výsledkem je 32-bitové dvojslovo, můžeme výpočet podobně jako u Fourierovy transformace provést *in-place*, tj. výstupními hodnotami postupně přepisujeme vstupní hodnoty, které již nebudou potřeba.

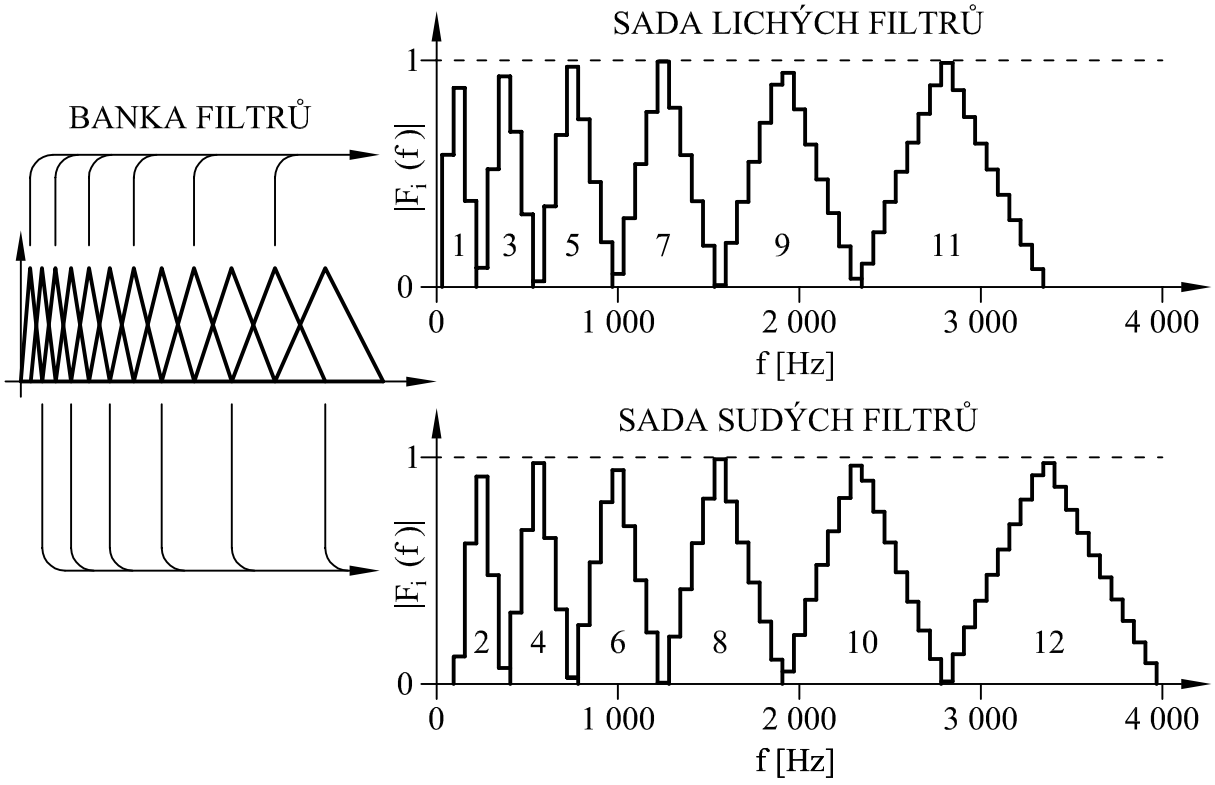
4.4.4. Banka filtrů

Připomeňme, že výpočet banky filtrů spočívá ve vyčíslení skalárních součinů vektoru, který jsme získali v předchozím kroku, s několika váhovými vektory, které odpovídají přenosovým charakteristikám filtrů. Tuto operaci by bylo možno elegantně zapsat pomocí maticového násobení. Příslušná matice by ovšem musela mít rozměr $64 \times 12 = 768$ prvků a zabírala by významné množství paměti; navíc většina jejích prvků by byla nulová.

Proto byl zvolen mnohem efektivnější postup: banku filtrů rozdělíme na liché a sudé filtry, jak je naznačeno na obrázku 4.5. Tím získáme dvě sady filtrů, jejichž charakteristiky se pak již nepřekrývají. Pak můžeme celou sadu filtrů popsat jediným váhovým vektorem, přičemž vstupní vektor procházíme pouze jedenkrát a vždy ve správný okamžik uložíme výstup právě vypočteného filtru a zahájíme výpočet dalšího filtru. Na obrázku 4.5 jsou znázorněny skutečné váhové vektory, které byly vytvořeny z trojúhleníkových charakteristik filtrů (podle obrázku 2.7). Každý prvek těchto vektorů určuje, jak výrazně se příslušná frekvence promítne do výstupu daného filtru.

Prvky váhového vektoru jsou uloženy jako 16-bitová slova, obsah jednotlivých frekvencí v rámci (který je výsledkem předcházejícího kroku) načteme ve formě 32-bitového dvojslova. Násobení je tedy nutno rozdělit do dvou kroků, neboť mikrokontrolér umí přímo násobit pouze dvě 16-bitová slova.

Výstupní hodnoty tohoto kroku výpočtu neukládáme, ale ihned je logaritmujeme a ukládáme až vypočtený logaritmus.



Obrázek 4.5: Rozdělení banky filtrů

4.4.5. Logaritmování

Knihovny prostředí *CodeWarrior* neobsahují funkci pro výpočet logaritmu v pevné čárce. Obsahují pouze funkci `log`, která slouží k výpočtu logaritmu v plovoucí čárce. Tato funkce je ovšem příliš pomalá (pro představu: při jejím použití by jen samotné logaritmování spotřebovalo přibližně 1/5 celkového dostupného výpočetního výkonu). Proto byla vytvořena vlastní funkce, jejíž princip je inspirován funkcí `log`, ale ve které probíhají veškeré výpočty v pevné čárce. Tento princip bude nyní popsán.

Při výpočtu MFCC principielně nezáleží na tom, jaký základ logaritmu použijeme. Namísto přirozeného logaritmu tedy můžeme (pro zjednodušení dalšího popisu) použít logaritmus o základu 2.

Vstupem funkce pro výpočet logaritmu bude kladné číslo v akumulátoru, tedy hodnota $x \in \langle 2^{-31}; 16 - 2^{-31} \rangle$. Pro tento rozsah vstupních hodnot nelze logaritmus uspokojivě nahradit polynomem. Využijeme tedy skutečnosti, že každé kladné číslo x lze převést do tvaru

$$x = m.2^e \tag{4.13}$$

kde $m \in \langle 0{,}5; 1)$ je *mantisa* a $e \in \mathbb{Z}$ je *exponent*. Pro tento převod disponují kontroléry řady 56800 speciální instrukcí `NORM`, takže jej lze provést snadno a rychle. Dvojkový logaritmus čísla x lze pak realizovat jako

$$\log_2(x) = \log_2(m.2^e) = \log_2(m) + \log_2(2^e) = \log_2(m) + e \tag{4.14}$$

Výpočet $\log_2(m)$ je mnohem jednodušší úloha, neboť na intervalu $\langle 0{,}5; 1)$ lze logaritmus s uspokojivou přesností aproximovat polynomem 3. řádu

$$\log_2(m) \approx \sum_{i=0}^{3} a[i].m^i \tag{4.15}$$

kde koeficienty polynomu $a[i]$ byly stanoveny pomocí nástroje MATLABu *Basic Fitting*

$$\begin{aligned} a[0] &= -3{,}150\,735 & a[1] &= 6{,}079\,919 \\ a[2] &= -4{,}181\,657 & a[3] &= 1{,}252\,897 \end{aligned}$$

Pro výše uvedený interval vstupních hodnot vrací funkce $\log_2(x)$ hodnoty z intervalu $\langle -31; 4)$. Tento rozsah není možno vyjádřit pomocí 16-bitového slova (ve kterém má být nakonec výstupní hodnota uložena). Použijeme proto lineární transformaci a celkový výstup funkce y určíme s použitím vztahu (4.14) jako

$$y = k.\log_2(x) + b = k.\log_2(m) + k.e + b \tag{4.16}$$

kde k, b jsou parametry lineární transformace. Optimální využití rozsahu 16-bitového slova zajišťuje volba hodnot

$$k = \frac{1\,872}{32\,768} \qquad b = \frac{25\,278}{32\,768} \tag{4.17}$$

Pokud dosadíme vztah (4.15) do (4.16) a výraz upravíme, aby nedošlo k přetečení, obdržíme výsledný postup výpočtu

$$y = \left(\frac{a[0]}{a[1]} + m + \frac{a[2]}{a[1]}m^2 + \frac{a[3]}{a[1]}m^3\right).(k.a[1]) + k.e + b \tag{4.18}$$

Pokud se na vstupu funkce objeví hodnota $x = 0$, pro kterou není logaritmus definován, vrací funkce nejmenší možnou hodnotu, tj. $y = -1$.

4.4.6. Oprava měřítka

Jak bylo zmíněno v kapitole 4.4.2, funkce `rfft` může během výpočtu zpracovávaný signál několikrát vydělit dvěma, aby se zabránilo přetečení. Tento jev musíme nyní kompenzovat, abychom obdrželi výsledek nezávislý na počtu provedených dělení.

Pokud označíme výstupní vektor FFT jako $\mathbf{q}$ a výstupní vektor logaritmování jako $\mathbf{y}$, pak lze postup, popsaný v předchozích třech kapitolách, shrnout vztahem

$$\mathbf{y} = k.\log_2(\mathbf{F}.|\mathbf{q}|^2) + b \tag{4.19}$$

kde $\mathbf{F}$ je transformační matice, reprezentující banku filtrů. Pokud během výpočtu FFT bude jednou použito dělení dvěma, pak bude výstupem FFT vektor $\mathbf{q}/2$ a nový výstupní vektor logaritmování $\mathbf{y}^*$ bude

$$\begin{aligned} \mathbf{y}^* &= k.\log_2\left(\mathbf{F}.\left|\frac{\mathbf{q}}{2}\right|^2\right) + b = k.\log_2\left(\mathbf{F}.\frac{1}{4}|\mathbf{q}|^2\right) + b = \\ &= k.\left[\log_2\left(\frac{1}{4}\right) + \log_2\left(\mathbf{F}.|\mathbf{q}|^2\right)\right] + b = k.\log_2(\mathbf{F}.|\mathbf{q}|^2) + b - 2k \end{aligned} \tag{4.20}$$

Vidíme, že výsledky $\mathbf{y}$ a $\mathbf{y}^*$ se liší o člen $2k$. Proto musíme k výstupu logaritmování přičíst člen $2k$ tolikrát, kolikrát došlo během výpočtu FFT k dělení dvěma. To ovšem může způsobit saturaci při vysokých hodnotách $\mathbf{y}$. Lepší výsledky obdržíme, pokud naopak odečteme člen $2k$ tolikrát, kolikrát během výpočtu FFT nedošlo k dělení dvěma.

Pokud není na vstup zařízení přiveden zvukový signál, zpracováváme pouze slabý šum. V tomto případě obsahuje $\mathbf{y}$ poměrně malé hodnoty, které se rámec od rámce velmi liší, neboť se začíná výrazně uplatňovat vliv kvantování a zaokrouhlovacích chyb. Abychom tento jev potlačili, provedeme oříznutí prvků $y[i]$ vektoru $\mathbf{y}$ na úrovni y_{min}, čímž obdržíme nový vektor $\mathbf{y}_o$ s prvky $y_o[i]$, tedy

$$y_o[i] = \begin{cases} y[i] & \text{pro } y[i] \geq y_{min} \\ y_{min} & \text{pro } y[i] < y_{min} \end{cases} \tag{4.21}$$

Hodnotu y_{min} je nutno stanovit empiricky.

4.4.7. Diskrétní kosinová transformace

Vzhledem k tomu, že vstupem DCT je logaritmovaný výstup banky filtrů (tedy v našem případě vektor o délce jen 12 prvků), není nutno používat podobný „trik“, jaký byl uveden u FFT. Výpočet lze provést vpodstatě přímo dle definičního vztahu (2.15). Abychom se vyhnuli výpočtu kosinů za běhu programu, vyjádříme kosinovou transformaci pomocí transformační matice $\mathbf{T}$. Tím se výpočet kosinové transformace sloupcového vektoru $\mathbf{y}_o$ redukuje na maticové násobení

$$\mathrm{DCT}(\mathbf{y}_0) = \mathbf{T}.\mathbf{y}_o \tag{4.22}$$

přičemž transformační matici $\mathbf{T}$ určíme aplikací DCT na každý sloupec matice identity $\mathbf{I}$ o rozměrech 12×12

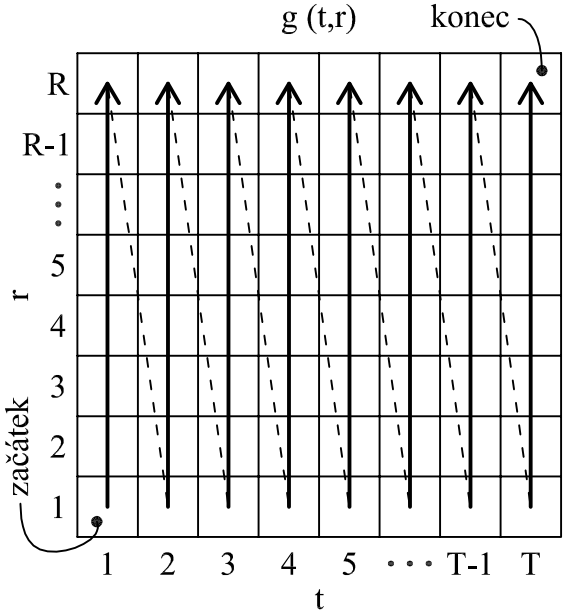
$$\mathbf{T} = \mathrm{DCT}(\mathbf{I}) \tag{4.23}$$

Výstupem DCT je vektor MFCC. Protože ale potřebujeme jen prvních 5 MFCC (zbytek pro rozpoznávání nevyužíváme), můžeme z matice $\mathbf{T}$ odstranit 6. až 12. řádek, takže pomocí maticového násobení přímo získáme pětici požadovaných MFCC. Dodejme, že v programu jsou prvky matice $\mathbf{T}$ 3,5krát zmenšeny, aby nemohlo dojít k přetečení.

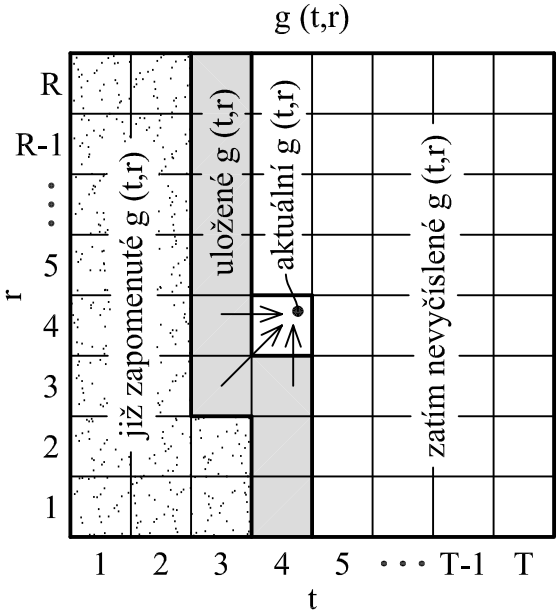
4.4.8. Dynamické borcení času

Největším problémem při implementaci DTW je uložení ceny cesty do každého bodu roviny $g(t, r)$ (vztah (2.21)). Uvažujme délku rozpoznávaného příkazu 1 s. Stejná bude i délka každého vzoru. Při vzorkovací frekvenci 8 kHz a posunu rámce 80 vzorků bude tedy rozpoznávaný příkaz i vzor reprezentován sekvencí 100 vektorů. Pak by matice, do které ukládáme $g(t, r)$, musela mít rozměr 100×100 prvků. Pro každý vzor navíc musíme vytvářet samostatnou matici, chceme-li vyhodnocovat všechny vzory najednou. Protože máme celkem k dispozici jen 4 kB datové paměti RAM, je sestavení celých matic nemožné.

K vyhodnocení podobnosti sekvencí vektorů naštěstí potřebujeme jen jediný prvek $g(T, R)$, jak plyne z (2.25). Navíc $g(t, r)$ závisí jen na třech sousedních prvcích, jak uvádí vztah (2.21). Pokud jednotlivé prvky matice, které odpovídají $g(t, r)$, budeme vyčíslovat po sloupcích dle schématu 4.6, vystačíme vždy s uložením části předešlého sloupce a části aktuálního sloupce, jak ukazuje obrázek 4.7. Je zřejmé, že takto stačí ukládat nejvýše $(R+1)$ hodnot, tedy v našem případě 101 hodnot. To již nepředstavuje z hlediska dostupné paměti žádný problém.



Obrázek 4.6: Pořadí výpočtu $g(t, r)$



Obrázek 4.7: Ukládání dat při výpočtu

Uvedený přístup je vhodný i z hlediska časového rozvržení výpočtu. Proměnná r totiž indexuje jednotlivé vektory vzoru, zatímco proměnná t indexuje jednotlivé vektory rozpoznávané sekvence. Výpočet dalšího sloupce tedy můžeme provést vždy, když získáme nový vektor parametrů, popisující aktuálně zpracovaný rámec rozpoznávaného zvuku. Sekvenci vektorů, která popisuje daný vzor, máme neustále celou k dispozici, neboť je uložena v paměti FLASH. Uveďme nyní celkový postup výpočtu DTW:

Inicializace – Při zpracování prvního sloupce bychom měli použít vztahy (2.22) a (2.24). Abychom nemuseli zpracování prvního sloupce programovat zvlášť, dodefinujeme fiktivní „nultý“ sloupec s hodnotami

$$g(0, r) = \infty \quad \text{pro } r \neq 1 \tag{4.24}$$

$$g(0, 1) = 0 \tag{4.25}$$

a tyto hodnoty uložíme do oblasti paměti, kam budeme později ukládat vypočtené $g(t, r)$. Pak můžeme i pro výpočet prvního sloupce použít vztahy (2.21) a (2.23), neboť minimum ve vztahu (2.21) nekonečnou hodnotu nikdy nevybere[1].

[1]Naznačeným postupem ve skutečnosti nedosáhneme naprosto přesně stejného efektu, jako s použitím vztahů (2.22) a (2.24). Odchylka se však týká jen prvku $g(1, 2)$ a je v praxi zanedbatelná.

Hodnoty $g(t, r)$ budeme ukládat pomocí 16-bitových slov. Proto nekonečno ve výrazu (4.24) nahradíme nejvyšší možnou hodnotou, tj. $(1 - 2^{-15})$. Obdobně nulu ve výrazu (4.25) nahradíme hodnotou -1, abychom využili i záporná čísla.

Výpočet – Vždy, když určíme MFCC aktuálního rámce, provedeme vyhodnocení jednoho sloupce. Počítáme postupně vzdálenosti mezi vektorem MFCC aktuálního rámce a všemi vektory MFCC vzoru – vztah (2.17). Uložené hodnoty $g(t, r)$ aktualizujeme dle výrazu (2.21). K výpočtu první hodnoty ve sloupci použijeme vztah (2.23). Výpočet probíhá do té doby, než zpracujeme požadovaný počet rámců (času 1 s odpovídá 100 rámců).

Výstup – Po ukončení výpočtu můžeme poslední vypočtenou hodnotu $g(T, R)$ považovat za výsledek, tj. za vzdálenost srovnávaných sekvencí vektorů. Pokud je délka všech vzorů stejná a délka rozpoznávaného úseku zvuku se také nemění, není ani nutné provádět normalizaci na jednotkovou délku sekvencí dle vztahu (2.25).

Připomeňme, že tento postup je třeba provádět nezávisle pro každý vzor. Algoritmus DTW tedy běží „paralelně“ v tolika instancích, kolik je rozpoznávaných vzorů.

Při výpočtu vzdálenosti dvou vektorů dle vztahu (2.17) se nevyhneme výpočtu odmocniny. K tomuto účelu byla použita hotová funkce `mfr32Sqrt`. Tato funkce využívá skutečnosti, že kontroléry řady 56800 sice neumí přímo vypočítat odmocninu, umí však efektivně počítat funkci inverzní, tj. druhou mocninu. Protože druhá odmocnina je na celém definičním oboru rostoucí, je k výpočtu odmocniny použit algoritmus postupné aproximace (půlení intervalu), jehož princip popisuje následující příklad – výpočet $\sqrt{0{,}09} = 0{,}3$:

1. Protože odmocňované číslo leží v intervalu $\langle 0; 1)$, musí i výsledek ležet v témže intervalu. Zkusíme tedy umocnit číslo ležící v polovině tohoto intervalu, tedy 0,5. Protože $0{,}5^2 = 0{,}25$ je větší než vstupní hodnota 0,09, leží hledaná odmocnina v intervalu $\langle 0; 0{,}5)$.

2. Vezměme nyní opět střed intervalu $\langle 0; 0{,}5)$, tj. číslo 0,25. Protože $0{,}25^2 = 0{,}062\,5$ je menší než vstupní hodnota 0,09, leží hledaná odmocnina v intervalu $\langle 0{,}25; 0{,}5)$.

3. V dalším kroku určujeme druhou mocninu čísla 0,375 atd., až se přiblížíme správné hodnotě s požadovanou přesností. U funkce `mfr32Sqrt` je výpočet ukončen tehdy, jsou-li správně určeny všechny bity výstupního 16-bitového slova.

Před výpočtem odmocniny je součet druhých mocnin rozdílů vydělen číslem 64 (což odpovídá dělení číslem 8 po odmocnění). To zajišťuje vhodné přizpůsobení hodnot $g(t, r)$ do reprezentovatelného rozsahu $\langle -1; 1)$. Hodnota 64 byla stanovena empiricky.

4.4.9. Vyhodnocení

Do vyhodnocení vstupují rozdíly mezi rozpoznávanou sekvencí a všemi vzorovými sekvencemi. Tyto rozdíly jsme v předchozím kroku stanovili pomocí DTW. Z těchto hodnot vybíráme minimum, tj. největší podobnost vzoru a rozpoznávaného zvuku. Na rozdíl od zkoušky algoritmu v MATLABu ale musíme při reálném nasazení počítat s možností, že uživatel řekl slovo, které se nepodobá ani jednomu vzoru. Proto nejmenší nalezený rozdíl navíc porovnáváme s určitou prahovou hodnotou. Pokud je rozdíl větší než prahová hodnota, prohlásíme výsledek za neplatný.

Nastavení prahové hodnoty je nutno provést empiricky. Z teoretického hlediska je výše uvedené porovnání statistickým testem hypotézy, zda uživatel vyslovil neplatný hlasový příkaz. U tohoto testu může docházet ke dvěma druhům chyb – buďto uživatel řekl platný příkaz, ale výsledek je prohlášen za neplatný, nebo uživatel neřekl platný příkaz, ale výsledek je přesto prohlášen za platný. Změnou prahové hodnoty vždy snížíme pravděpodobnost jedné chyby, zároveň však zvýšíme pravděpodobnost druhé chyby. Je proto nutno najít optimum, kde jsou obě chyby poměrně málo pravděpodobné, případně kdy mírně převažuje pravděpodobnost chyby s méně závažnými důsledky.

4.4.10. Detektor řečové aktivity

Při ověřování funkčnosti algoritmu v kapitole 2.7 jsme rozpoznávali předem připravené úseky signálu, z nichž každý obsahoval právě jedno slovo. V případě hlasového modulu je situace jiná: vstupem je souvislý zvukový signál o „nekonečné“ délce. Protože DTW umí porovnávat pouze sekvence konečné délky, musíme ze vstupního signálu izolovat úsek, obsahující rozpoznávané slovo. Pokud pevně stanovíme délku rozpoznávané sekvence např. na 1 s, stačí pak identifikovat začátek sekvence.

Úlohu identifikace začátku sekvence lze v zásadě řešit dvěma způsoby. První způsob spočívá v použití tlačítka, jehož stisknutím uživatel zahájí proces rozpoznávání. Tento způsob má zjevnou nevýhodu v tom, že principielně narušuje „bezdotykovost“ hlasového ovládání. Tato nevýhoda je však vyvážena absolutní spolehlivostí této techniky a proto se aktivační tlačítko u jednodušších systémů používá.

Druhý způsob je založen na použití detektoru řečové aktivity. Detektor řečové aktivity na základě výpočtu některé charakteristiky vstupního signálu odhaduje, zda do hlasového modulu vstupuje užitečný řečový signál nebo pouze šum. Proces rozpoznávání pak spustíme tehdy, když detektor zjistí přítomnost řeči ve vstupním signálu. U této metody však může docházet jak k nechtěnému spuštění, tak ke stavům, kdy rozpoznávání nebylo spuštěno i přesto, že na vstupu je řečový signál.

Do software hlasového modulu byly implementovány obě výše popsané metody. Během provozu pak volíme jednu z těchto metod pomocí přepínače, jak bude popsáno v kapitole 4.6. Detektor řečové aktivity, který je použit v hlasovém modulu, měří energii ve vstupním signálu a porovnává ji s určitou prahovou hodnotou. Tento princip vychází z předpokladu, že energie užitečného hlasového signálu je výrazně vyšší než energie zvuko-

vého pozadí. Měření energie probíhá zvlášť pro každý rámec, a to ještě před aplikováním okna. Pro výpočet energie rámce E je užit vztah, převzatý z [2]

$$E = \sum_{n=0}^{l_{ram}-1} (x[n])^2 \tag{4.26}$$

kde $x[n]$ představuje jednotlivé vzorky rámce o délce l_{ram}. Tato metoda není tak spolehlivá, jako některé pokročilejší algoritmy, je však implementačně jednoduchá a rychlá.

4.5. Komunikace s nadřazeným systémem

Komunikace s nadřazeným systémem nebyla do software hlasového modulu implementována. V době psaní této práce totiž nebylo rozhodnuto o použití hlasového modulu v konkrétní aplikaci. Hlasový modul je však možno ovládat pomocí tlačítka a přepínačů, jak bude popsáno v následující kapitole. Výsledek rozpoznávání se zobrazuje na sedmisegmentovém displeji.

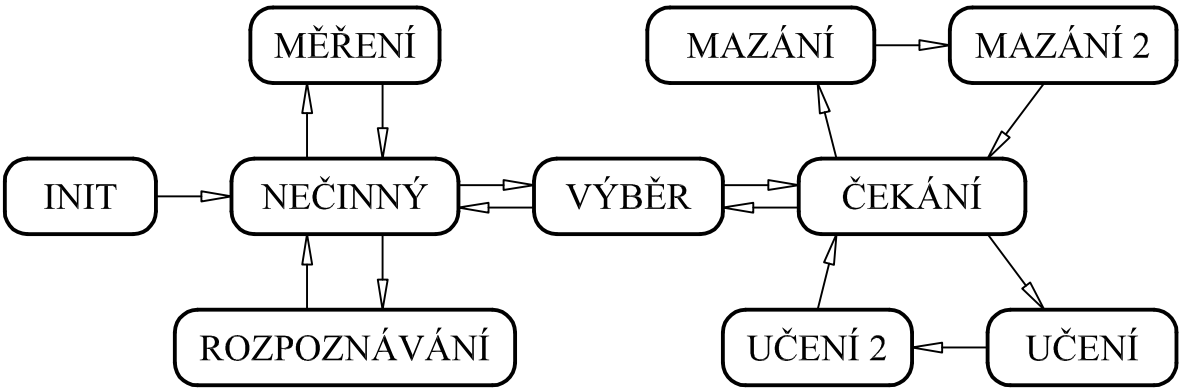
4.6. Stavový automat

V kapitole 4.4 byla podrobně popsána činnost hlavního programu v režimu rozpoznávání. Tento režim je bezesporu nejdůležitější, kromě toho však hlasový modul musí být schopen záznamu nových vzorů. Je také vhodné, aby hlasový modul uměl zobrazit na displeji úroveň vstupního signálu, abychom mohli správně nastavit zesílení analogové vstupní části. Pro přehledné a deterministické řízení všech popisovaných funkcí je vhodné použít stavový automat, jehož činnost ovládá uživatel pomocí tlačítka a přepínačů.

Stavý automat je umístěn v hlavním programu za výpočtem MFCC. Je to z toho důvodu, že MFCC je třeba vypočítat jak v režimu rozpoznávání, tak v režimu učení (jak bylo ukázáno již na obrázku 2.2). Naopak DTW či ukládání vzorů je závislé na aktuálně zvoleném režimu a tyto funkce jsou tedy dle potřeby volány stavovým automatem.

Obrázek 4.8 zobrazuje stavy automatu a možné přechody mezi nimi. Po zapnutí se automat nachází ve stavu *INIT*. V tomto stavu setrvá asi 1 s. Tato doba je určena k ustálení analogové části modulu po připojení napájení. Na displeji je přitom zobrazována jednoduchá animace. Po uplynutí časového intervalu automat přejde do stavu *NEČINNÝ*. Přechod do ostatních stavů koresponduje s jednotlivými režimy hlasového mudulu:

1. Ve stavu *ROZPOZNÁVÁNÍ* probíhá rozpoznávání zvukového vstupu. Do tohoto stavu se lze dostat buď stiskem tlačítka nebo signálem z detektoru řečové aktivity. O způsobu aktivace rozpoznávání rozhoduje čtvrtý přepínač – pokud je v poloze vypnuto, spouštíme rozpoznávání tlačítkem; pokud je v poloze zapnuto, používá se detektor řečové aktivity. Během rozpoznávání svítí na displeji pouze desetinná tečka. Stav *ROZPOZNÁVÁNÍ* je ukončen po zpracování požadované délky zvukového vstupu a automat přejde zpět do stavu *NEČINNÝ*. Při tom se na displeji zobrazí číslo rozpoznaného vzoru.



Obrázek 4.8: Stavový automat

2. Stav *MĚŘENÍ* slouží k zobrazení úrovně vstupního signálu na displeji. Do tohoto stavu se lze dostat zapnutím prvního přepínače, po vypnutí přepínače se automat vrací zpět do stavu *NEČINNÝ*. Úroveň signálu je určována v obsluze přerušení AD převodníku (ještě před digitálním filtrem) měřením maxima absolutní hodnoty vstupního signálu. Segmenty na obvodu displeje se rozsvěcují ve směru hodinových ručiček, počet svítících segmentů odpovídá špičkové úrovni vstupního signálu. Tím je napodoben efekt klasického indikátoru vybuzení typu „bargraf“. Maximální úrovni signálu odpovídá šest rozsvícených segmentů (celý obvod displeje). Pokud je na vstupu ticho nebo velmi slabý signál, klesá počet svítících segmentů postupně k nule.

3. Stav *VÝBĚR* je výchozím bodem pro nahrávání a výmaz vzorů. Do tohoto stavu automat přejde při zapnutí druhého přepínače, vypnutím přepínače se lze dostat zpět do stavu *NEČINNÝ*. Ve stavu *VÝBĚR* je na displeji zobrazeno číslo aktuálně zvoleného vzoru, mezi vzory se lze přepínat stiskem tlačítka. Desetinná tečka svítí v případě, že aktuálně vybraný vzor obsahuje platná data a lze jej použít k rozpoznávání. Pokud desetinná tečka nesvítí, je aktuální vzor prázdný.

   Při zapnutí třetího přepínače přejde automat do stavu *ČEKÁNÍ*, vypnutím třetího přepínače se lze vrátit zpět do stavu *VÝBĚR*. Ve stavu *ČEKÁNÍ* je zobrazeno číslo vzoru s informací o jeho využití, stejně jako ve stavu *VÝBĚR*. Pokud ve stavu *ČEKÁNÍ* stiskneme tlačítko a aktuálně vybraný vzor obsahuje platná data, dojde k přechodu do stavu *MAZÁNÍ*, ve kterém je aktuální vzor vymazán z paměti. Pak přejde automat do stavu *MAZÁNÍ 2*, ve kterém je na displeji zobrazena „diagonální“ čára jako symbol škrknutí, výmazu aktuálního vzoru. Asi po sekundě se automat vrací do stavu *ČEKÁNÍ*. Pokud ve stavu *ČEKÁNÍ* stiskneme tlačítko a aktuálně vybraný vzor neobsahuje platná data (tj. je prázdný), přejde automat do stavu *UČENÍ*. V tomto stavu probíhá záznam nového vzoru. Jakmile je vzor zaznamenán, přejde automat do stavu *UČENÍ 2*, ve kterém je na displeji zobrazeno zatržítko na znamení toho, že nový vzor byl uložen. Asi po sekundě se automat vrací do stavu *ČEKÁNÍ*.

5. Vyhodnocení

5.1. Využití prostředků kontroléru 56F805

Až doposud jsme se nezabývali otázkou, kolik vzorů je hlasový modul schopen rozpoznat. Počet vzorů je totiž omezen možnostmi kontroléru, konkrétně

1. výpočetním výkonem, neboť pro každý vzor musí běžet nezávislé DTW
2. množstvím paměti FLASH, kterou používáme pro uložení samotných vzorů
3. množstvím paměti RAM, kterou používáme pro uložení $g(t, r)$

Proto byla volba počtu vzorů provedena až po dokončení implementace software, kdy již byl znám konkrétní vliv těchto omezujících faktorů.

Pokud bychom samotné vzory ukládali do datové paměti FLASH (což je nejlogičtější řešení), pak by byl počet vzorů limitován právě velikostí této paměti a nemohli bychom použít více než 7 vzorů. Proto jsou vzory uloženy v programové paměti FLASH. Tato paměť je primárně určena k uložení programu (nikoliv uživatelských dat), proto je přístup k takto uloženým vzorům mírně pomalejší a komplikovanější. Na druhou stranu je takto možno zvýšit počet vzorů až na 12, kdy začne být omezujícím faktorem množství datové paměti RAM. Jednotlivé vzory jsou na displeji hlasového modulu reprezentovány číslicemi „0“ až „9“ a písmeny „A“ a „b“, písmeno „F“ značí neúspěšné rozpoznání.

Využití prostředků kontroléru v režimu rozpoznávání pro 12 vzorů je shrnuto v tabulce 5.1. Vidíme, že většina výpočetního výkonu se spotřebuje na výpočet DTW, zbytek je zanedbatelný. Tento výsledek není překvapující, uvědomíme-li si, že výpočet DTW pro 12 vzorů zahrnuje výpočet Euklidovské vzdálenosti celkem 120 000 dvojic pětiprvkových vektorů, který proběhne za dobu 1 s. Vzory, uložené v programové paměti FLASH, paradoxně využívají větší část této paměti než samotný software. Datová paměť RAM je při 12 vzorech využita téměř celá, neboť počet vzorů byl volen právě dle velikosti této paměti. Naopak datová paměť FLASH je využita jen minimálně.

Hodnoty využití jednotlivých pamětí v tabulce 5.1 byly získány z paměťové mapy, která je vedlejším produktem překladu (respektive linkování) software. Uvedené hodnoty platí pro konfiguraci software, kdy je datová paměť RAM inicializována z datové paměti FLASH. Spotřeba výpočetního výkonu byla měřena praktickým experimentem. Při měření bylo využito té skutečnosti, že pokud kontrolér neprovádí žádný užitečný výpočet, nachází se v čekací smyčce na začátku hlavního programu (popsáno v kapitole 4.4.1). Tuto smyčku opustí až tehdy, když je připraven nový rámec. Pokud zjistíme počet průchodů smyčkou, lze z tohoto počtu odvodit počet instrukčních cyklů, který kontroléru zbyl po dokončení všech výpočtů. Z počtu těchto instrukčních cyklů lze pak snadno určit vytížení kontroléru. Pokud pak vyřadíme některou část software z činnosti, lze snadno zjistit, jakou část výpočetního výkonu zabírala.

Výpočetní výkon		
- k dispozici	40,0 MIPS	
- využito celkem	26,8 MIPS	(67%)
z toho - příjem vzorků z ADC, digitální filtr	2,2 MIPS	(5%)
- FFT rámce	0,8 MIPS	(2%)
- DTW pro 12 vzorů	23,6 MIPS	(59%)
- ostatní	0,2 MIPS	(1%)
Programová paměť FLASH		
- k dispozici	63,0 kB	
- využito celkem	17,3 kB	(27%)
z toho - software	5,3 kB	(8%)
- 12 vzorů	12,0 kB	(19%)
Datová paměť RAM		
- k dispozici	4,0 kB	
- využito celkem	3,8 kB	(95%)
z toho - paměť vzorků	0,5 kB	(12%)
- stav DTW pro 12 vzorů	2,3 kB	(58%)
- ostatní	1,0 kB	(25%)
Datová paměť FLASH		
- k dispozici	8,0 kB	
- využito celkem	0,9 kB	(11%)

Tabulka 5.1: Využití kontroléru

5.2. Knihovna *VRLite-1*

Pro účely rozpoznávání izolovaných slov na signálových kontrolérech řady 56800 existuje hotová knihovna *VRLite-1*. Pro účely hlasového modulu nepřichází využití této knihovny v úvahu, neboť se jedná se o komerční produkt, jehož licence stojí 1 500 $. Můžeme ovšem využít volně dostupné informace o této knihovně [12], [13] a použít je pro zhodnocení výpočetní a paměťové náročnosti hlasového modulu.

Knihovna *VRLite-1* zajišťuje zpracování rámců, rozpoznávání a učení vzorů. Digitalizaci vstupního signálu musí řešit uživatel této knihovny, neboť tato část silně závisí na konkrétním použitém hardware. Knihovna *VRLite-1* se skládá ze dvou částí. První část, tzv. *front-end*, zajišťuje výpočet parametrů rámce a pracuje v reálném čase. Činnost této části je dle [12] založena na blíže nespecifikované bance filtrů. Je tedy možné, že se jedná o podobný výpočet MFCC, jaký byl popsán v kapitole 2.5. Druhá část, tzv. *back-end*, provádí samotné učení a rozpoznávání vzorů a nepracuje v reálném čase, ale je volána až po ukončení zvukového vstupu. Tato část je založena na tzv. *skrytých Markovových modelech*, což je nástroj založený na pravděpodobnostním modelování sekvencí [2].

Výsledky porovnání knihovny *VRLite-1* a hlasového modulu shrnuje tabulka 5.2. Protože část knihovny *VRLite-1* nepracuje (na rozdíl od hlasového modulu) v reálném čase,

byly některé údaje z [13] za účelem srovnání přepočítány tak, jako by knihovna v reálném čase pracovala, přičemž byla uvažována délka rozpoznávaného slova 1 s. Ze software hlasového modulu byly uvažovány jen ty části, které bezprostředně souvisí s rozpoznáváním (nebylo započteno čtení vzorků, stavový automat apod.), aby bylo srovnání relevantní.

vlastnost	*VRLite-1*	hlasový modul
Potřebný výpočetní výkon		
- výpočet parametrů rámců (*front-end*/MFCC)	28,0 MIPS	1,0 MIPS
- rozpoznávání jednoho vzoru (*back-end*/DTW)	0,6 MIPS	2,0 MIPS
Paměťová náročnost		
- konstanty pro výpočet	1,1 kB	0,8 kB
- pomocný prostor pro mezivýpočty	9,1 kB	0,3 kB[1]
- uložení jednoho vzoru	0,2 kB	1,0 kB
- vlastní software	13,9 kB	0,5 kB

Tabulka 5.2: Porovnání knihovny *VRLite-1* a hlasového modulu

Porovnejme nejprve potřebný výpočetní výkon. Z tabulky 5.2 plyne, že výpočet parametrů rámce je u hlasového modulu výrazně méně náročný. Hlasový modul totiž používá pouze základní variantu MFCC. Knihovna *VRLite-1* pravděpodobně používá sofistikovanější metodu, která je výpočetně náročnější, ale zřejmě poskytuje lepší výsledky. Rozpoznávání vyžaduje menší výpočetní výkon u knihovny *VRLite-1*.

Ohledně paměťové náročnosti lze říci, že obě metody používají přibližně stejný objem konstant. Hlasový modul vyžaduje výrazně menší pomocný prostor pro mezivýpočty. Tento prostor je však potřeba navýšit pro každý další rozpoznávaný vzor, neboť u hlasového modulu probíhá rozpoznávání „paralelně“. U knihovny *VRLite-1* je velikost pomocného prostoru na počtu vzorů nezávislá, neboť vyhodnocování vzorů probíhá postupně.

Uložení jednoho vzoru vyžaduje u *VRLite-1* oproti hlasovému modulu jen asi pětinu paměti. *VRLite-1* totiž zřejmě každý vzor ukládá v „zabalené“ formě, kdy může být v jednom datovém slovu uloženo i více hodnot s nižší přesností (s menším počtem platných bitů). Naopak délka přeloženého kódu je výrazně nižší u hlasového modulu, neboť ten používá relativně jednoduchý algoritmus bez jakýchkoliv přídavných vylepšení.

Tabulku 5.2 lze shrnout tvrzením, že základní požadavky software hlasového modulu jsou malé, prudce však rostou s každým dalším přidaným vzorem. To dobře koresponduje s filozofií algoritmu DTW, který je poměrně jednoduchý a úlohu řeší spíše „hrubou silou“. U knihovny *VRLite-1* jsou naopak základní požadavky poměrně vysoké, přidání několika vzorů však celkové nároky příliš neovlivní. Rozhodujícím parametrem pro srovnání obou metod by mohla být úspěšnost rozpoznávání (zejména v prostředí s velkým hlukem nebo šumem pozadí), tu však nebylo možno otestovat, neboť knihovna *VRLite-1* nebyla k dispozici. Lze však odhadnout, že z hlediska úspěšnosti rozpoznávání by knihovna *VRLite-1* dopadla výrazně lépe. Samostatné testování úspěšnosti rozpoznávání hlasového modulu popisuje následující kapitola.

[1] K této hodnotě musíme navíc připočíst 0,2 kB pro každý vzor.

5.3. Testování úspěšnosti rozpoznávání

Pro účely testování byla do hlasového modulu zaznamenána sada deseti vzorů. Kvůli přehlednosti byly jako vzory zvolena slova „nula“, „jedna“, „dva“, „tři“ atd. až „devět“. Povšimněme si, že jako vzor bylo zvoleno slovo „dva“, nikoliv „dvě“, abychom minimalizovali možnost záměny se slovem „pět“, které zní velmi podobně. Vzory byly do hlasového modulu uloženy několik dní před samotným testem, aby se mohly projevit drobné změny v hlase, ke kterým dochází u každého člověka vlivem aktuální nálady apod.

Dále byl pro test vytvořen seznam slov, která budeme rozpoznávat. Seznam obsahuje 9 výskytů každého vzoru. Pořadí slov v seznamu je náhodné. Navíc bylo do seznamu přidáno pět slov (každé ve dvou výskytech), které neodpovídají ani jednomu vzoru. Jedná se o slova „dvacítka“, „schůzka“, „něco“, „je“ a „výrok“. Na tato slova by měl hlasový modul reagovat neúspěšným rozpoznáním (blíže v kapitole 4.4.9). Seznam tedy obsahuje celkem 100 slov.

Výsledky testování samozřejmě ovlivňuje nastavení prahových hodnot algoritmu. Hodnota y_{min} pro oříznutí logaritmovaných výstupů banky filtrů (kapitola 4.4.6) byla nastavena na $-0{,}4$. Jako prahová energie rámce pro detektor řečové aktivity (kapitola 4.4.10) byla zvolena hodnota 0,25. Rozdíl porovnávaných sekvencí, od kterého je rozpoznávání prohlášeno za neplatné (kapitola 4.4.9), byl nastaven na 0,6. Během záznamu vzorů i během samotného testování byl mikrofon vzdálen přibližně 30 cm od úst mluvčího. Byla použita relativně hlasitá výslovnost s lehce zdůrazněnou artikulací.

5.3.1. Testování v klidném prostředí, aktivace tlačítkem

Nejprve bylo testování uskutečněno v prostředí se zanedbatelným zvukovým pozadím (uzavřená místnost bytu panelového domu). Proces rozpoznávání byl před každým slovem spouštěn pomocí tlačítka. Při tomto testu hlasový modul správně klasifikoval 88 slov z celkového počtu 100 slov. Z 90 slov, které odpovídaly některému ze vzorů, bylo správně rozpoznáno 84 slov, což odpovídá dokonce 93% úspěšnosti rozpoznávání. Z 10 slov, které neodpovídají ani jednomu vzoru, vedly 4 slova na neúspěšné rozpoznání a zbylých 6 slov bylo mylně přiřazeno k některému vzoru. Chování hlasového modulu při aktivaci tlačítkem v prostředí bez rušivých zvuků lze tedy označit za velmi dobré.

5.3.2. Testování v prostředí s rušivými zvuky, aktivace tlačítkem

V druhé fázi byla otestována odolnost hlasového modulu proti rušivým zvukům. Hlasový modul byl umístěn u otevřeného okna, ústícího do středně rušné ulice. Díky tomu byl hlas mluvčího smíchán s hlukem od projíždějících automobilů a s ptačím zpěvem. V tomto případě bylo správně klasifikováno jen 35 slov z celkového počtu 100 slov. Ve 35 případech skončilo rozpoznávání neúspěšně i u slov, která náleží k některému vzoru. Ve zbylých 30 případech bylo slovo přiřazeno k nesprávnému vzoru. Úspěšnost rozpoznávání v prostředí s rušivými zvuky je tedy špatná.

5.3.3. Testování v klidném prostředí, automatická aktivace

Dále byl hlasový modul testován opět v prostředí bez rušivých zvuků, ale s tím rozdílem, že rozpoznávání bylo spouštěno detektorem řečové aktivity. V tomto případě bylo správně klasifikováno jen 18 slov z celkového počtu 100 slov. V 73 případech skončilo rozpoznávání neúspěšně, i když dané slovo náleželo k některému vzoru. Ve zbylých 9 případech bylo slovo přiřazeno k nesprávnému vzoru. Úspěšnost rozpoznávání je tedy při použití detektoru řečové aktivity velmi špatná.

Velmi malou úspěšnost si lze v tomto případě vysvětlit tak, že pozdní reakce detektoru řečové aktivity má za následek potlačení začátku rozpoznávaného slova. Detektor řečové aktivity totiž reaguje až na určitou úroveň energie v signálu a začátek slova nemusí vždy této úrovně dosáhnout. Problémem může být i to, že všechny nahrané vzory začínají krátkým úsekem ticha. Pokud rozpoznávaná sekvence obsahuje hned na začátku řečový signál, nenalezne DTW v rozpoznávané sekvenci odpovídající tichý úsek a výsledkem je velká odchylka sekvence od vzoru.

Řešení výše naznačeného problému by mohlo spočívat v tom, že bychom i při aktivaci nahrávání vzorů použili detektor řečové aktivity. Pak by vzory i rozpoznávané sekvence byly oříznuté stejným způsobem, čímž by se úspěšnost rozpoznávání mohla zlepšit. Tento přístup by ale selhal v případě, že bychom takto zaznamenané vzory použili pro rozpoznávání s aktivací tlačítkem. Pak by totiž rozpoznávaná sekvence obsahovala oproti vzoru vždy něco navíc a úspěšnost rozpoznávání by zřejmě opět nebyla dobrá. Nabízí se ještě to řešení, že by existovaly dvě nezávislé sady vzorů (pro automatickou aktivaci a pro aktivaci tlačítkem). Toto řešení by ale znamenalo snížení počtu rozpoznávaných vzorů na polovinu a nebylo by tedy příliš praktické.

5.3.4. Testování v klidném prostředí, aktivace tlesknutím

Výše zmiňovaný problém se špatnou úspěšností rozpoznání při použití detektoru řečové aktivity byl nakonec vyřešen poměrně jednoduchým způsobem. Pokud těsně před tím, než řekneme rozpoznávané slovo, slabě tleskneme, tak toto tlesknutí aktivuje rozpoznávání. Rozpoznávání slova pak probíhá včetně úvodního ticha (mezi tlesknutím a slovem) a začátku slova. S tímto vylepšením bylo dosaženo (opět v klidném prostředí) úspěšné klasifikace 80 slov ze 100 slov. V 7 případech došlo k neúspěšnému rozpoznání, i když dané slovo náleželo k některému vzoru. V 13 případech pak skončilo rozpoznávání chybným přiřazením. Z uvedených hodnot plyne, že pokud rozpoznávání spouštíme tlesknutím, dosahuje hlasový modul jen mírně horší úspěšnosti, než při aktivaci tlačítkem.

V prostředí s rušivými zvuky nemělo smysl rozpoznávání s detektorem řečové aktivity testovat, neboť při daném nastavení a úrovni hluku docházelo neustále k nechtěné aktivaci rozpoznávání.

6. Závěr

Cílem této práce bylo vytvořit hlasový modul, který by umožnil ovládání nadřazeného systému (např. mobilního robotu) pomocí několika hlasových příkazů. Nejprve byl proveden teoretický rozbor základní metody pro rozpoznávání izolovaných hlasových příkazů. Abychom ověřili použitelnost této metody, byla provedena její implementace v prostředí MATLAB. Zkouška metody na sadě testovacích slov dopadla velmi dobře – algoritmus se během přiřazování 200 slov k 5 vzorům dopustil pouze jediné chyby.

Na základě odhadu náročnosti algoritmu byl jako hlavní prvek hardware hlasového modulu zvolen signálový kontrolér 56F805. Podle této volby pak byly navrženy ostatní části hlasového modulu. Značné úsilí bylo věnováno zejména návrhu antialiasingového filtru. Výsledkem je realizace hlasového modulu na jedné desce plošných spojů o rozměrech 73 × 66 mm. Hardware hlasového modulu splňuje všechny požadavky zadání. Celkový odběr hlasového modulu nepřesahuje 0,2 A a bylo by možné jej dále snížit využitím úsporného režimu kontroléru.

Návrh software byl veden snahou o co nejlepší využití možností kontroléru 56F805. Proto byla podstatná část software vytvořena v asembleru. Vzhledem k tomu, že při všech výpočtech je používána reprezentace čísel s pevnou řádovou čárkou, spočívá návrh software z velké části v analýze možnosti přetečení. V místech výpočtu, kde hrozí přetečení, byly zavedeny vhodné úpravy rozsahu hodnot.

Protokol pro komunikaci s nadřazeným systémem nebyl zatím implementován; tuto část bude vhodnější doplnit až poté, co bude rozhodnuto o konkrétním použití hlasového modulu. Hardware pro komunikaci s nadřazeným systémem po sběrnici RS-485 je na desce připraven. Hlasový modul je možno plně ovládat pomocí přepínačů a tlačítka (včetně nahrávání nových vzorů), výsledek rozpoznávání se zobrazuje na vestavěném displeji.

Prostředky kontroléru 56F805 umožňují současné použití až 12 vzorů, což je dostatečný počet. Výpočetní výkon kontroléru je během rozpoznávání využit z 67%, což ukazuje na správnou volbu kontroléru pro danou aplikaci – kontrolér není poddimenzován ani zbytečně předimenzován.

Praktická zkouška ukázala, že hlasový modul je schopen rozpoznávat příkazy s úspěšností až 93%. Tato hodnota platí v prostředí se zanedbatelným obsahem rušivých zvuků při aktivaci rozpoznávání tlačítkem. Při aktivaci rozpoznávání tlesknutím je úspěšnost o něco nižší, stále však vyhovující. V prostředí s výrazným obsahem akustického rušení je hlasový modul zatím takřka nepoužitelný.

V oblasti mobilní robotiky (pro kterou byl hlasový modul původně navrhován) zatím nelze použití hlasového modulu doporučit. Algoritmus, který hlasový modul používá, totiž neobsahuje žádnou metodu na potlačení zvuků v pozadí. Přesto však jistě existují aplikace, ve kterých by bylo možné hlasový modul v současném stavu využít.

Literatura

[1] *NEWTONDictate : Diktování do počítače* [online]. NEWTON Technologies [cit. 2011-05-07]. Dostupné z WWW: <http://www.diktovani.cz>.

[2] ČERNOCKÝ, Jan. *Zpracování řečových signálů : studijní opora* [online]. 2006-12-06 [cit. 2011-05-07]. Dostupné z WWW: <http://www.fit.vutbr.cz/study/courses/ZRE/public/opora/zre_opora.pdf>.

[3] *56F805 Data Sheet : Preliminary Technical Data* [online]. Rev. 16. Freescale Semiconductor, 09/2007 [cit. 2011-05-07]. 56 s. Dostupné z WWW: <http://cache.freescale.com/files/dsp/doc/data_sheet/DSP56F805.pdf>.

[4] *DSP56800 User Manual* [online]. Rev. 8. Freescale Semiconductor, 03/2007 [cit. 2011-05-07]. 702 s. Dostupné z WWW: <http://cache.freescale.com/files/dsp/doc/user_guide/DSP56F801-7UM.pdf>.

[5] *MAX9867 : Ultra-Low Power Stereo Audio Codec* [online]. Rev. 2. Maxim Integrated Products, 06/2010 [cit. 2011-05-07]. 55 s. Dostupné z WWW: <http://datasheets.maxim-ic.com/en/ds/MAX9867.pdf>.

[6] BELZA, Jaroslav. *Operační zesilovače pro obyčejné smrtelníky.* 1. vydání. Praha: BEN – technická literatura, 2004. Aktivní filtry, s. 46. ISBN 80-7300-115-2.

[7] *56F805 Evaluation Module User Manual* [online]. Rev. 5. Freescale Semiconductor, 07/2005 [cit. 2011-05-07]. 84 s. Dostupné z WWW: <http://cache.freescale.com/files/dsp/doc/user_guide/DSP56F805EVMUM.pdf>.

[8] *DSP56800 Family Manual* [online]. Rev. 3.1. Freescale Semiconductor, 11/2005 [cit. 2011-05-07]. 448 s. Dostupné z WWW: <http://cache.freescale.com/files/dsp/doc/ref_manual/DSP56800FM.pdf>.

[9] *Nápověda programu Matlab R2007a : IIR Polyphase Filter Design.* The MathWorks, 2006.

[10] *Freescale DSP Function Library : Component DSP_Func_DFR – Programmer's Guide.* Freescale Semiconductor, 2006.

[11] COCHRAN, William T., et al. *What is the Fast Fourier Transform?*. In *IEEE Transactions on Audio and Electroacoustics*, vol. au-15, no. 2, 1967. s. 45-55.

[12] *Embedded SDK (Software Development Kit) : Voice Recognition (VRLite-1) Library* [online]. Rev. 2. Motorola, 07/2002 [cit. 2011-05-07]. 56 s. Dostupné z WWW: <http://cache.freescale.com/files/dsp/doc/user_guide/SDK129.pdf>.

[13] *Targeting Freescale DSP56F80x Platform : Algorithm Performance Data.* Freescale Semiconductor.

Seznam zkratek

AD	*Analog-to-Digital*, analogově-digitální
ADC	*Analog-to-Digital Converter*, analogově-digitální převodník
CAN	*Controller Area Network*, druh sériové sběrnice
DCT	*Discrete Cosine Transform*, diskrétní kosinová transformace
DFT	*Discrete Fourier Transform*, diskrétní Fourierova transformace
DIP	*Dual In-line Package*, pouzdro s dvěma řadami vývodů
DTW	*Dynamic Time Warping*, dynamické borcení času
FFT	*Fast Fourier Transform*, rychlá Fourierova transformace, algoritmus pro výpočet DFT
FIFO	*First In First Out*, druh paměťové struktury, která při zápisu a čtení nemění pořadí prvků
FIR	*Finite Impulse Response*, skupina filtrů s konečnou délkou impulzní odezvy
GND	*Ground*, zemnící potenciál
IIR	*Infinite Impulse Response*, skupina filtrů s nekonečnou délkou impulzní odezvy
JTAG	*Joint Test Action Group*, rozhraní pro programování integrovaných obvodů a testování desek plošných spojů
LED	*Light Emitting Diode*, dioda vyzařující světlo
LPT	*Line Print Terminal*, paralelní port počítače
LQFP	*Low-profile Quad Flat Package*, druh zapouzdření integrovaných obvodů
MAC	*Multiply-Accumulate*, instrukce pro přičtení součinu k mezivýsledku nebo jednotka, realizující tuto instrukci
MFCC	*Mel-Frequency Cepstral Coefficients*, Mel-frekvenční cepstrální koeficienty
MIPS	*Million Instructions Per Second*, milión instrukcí za sekundu (jednotka výpočetního výkonu)
OnCE	*On Chip Emulation*, rozhraní pro ladění programu v cílovém mikrokontroléru
PWM	*Pulse Width Modulation*, pulzně-šířková modulace
RAM	*Random Access Memory*, paměť s náhodným přístupem; lze do ní libovolně elektricky zapisovat, ale při odpojení napájení se obsah ztratí
SPI	*Serial Peripheral Interface*, druh sériové sběrnice
TTL	*Transistor-Transistor Logic*, druh logických integrovaných obvodů, označení napěťové úrovně
UART	*Universal Asynchronous Receiver/Transmitter*, rozhraní pro sériovou komunikaci, druh sériové sběrnice
USB	*Universal Serial Bus*, druh sériové sběrnice

Seznam symbolů

$\|\|$	paralelní spojení
$\|\ldots\|$	absolutní hodnota
$\lfloor\ \rfloor$	zaokrohlení směrem dolů (tedy směrem k $-\infty$)
$\star$	konvoluce
a, b, c	koeficienty jmenovatele v přenosu $F(p)$
$a[i]$	koeficienty polynomu
$b[i]$	hodnota i-tého bitu
C	kapacita kondenzátoru
$C_x[n]$	cepstrum diskrétního signálu $x[n]$
$C_x(t)$	cepstrum spojitého signálu $x(t)$
$d(\ldots,\ldots)$	vzdálenost
e	exponent
e	Eulerovo číslo
E	energie rámce
$f(t)$	budící signál
f	frekvence
f_N, f_N^*	Nyquistovy frekvence, příslušející k f_S a f_S^*
f_S	vzorkovací frekvence
f_S^*	zvýšená vzorkovací frekvence
$F_i(f)$	přenos i-tého filtru z banky filtrů
$F(p)$	přenos analogového filtru
$\mathbf{F}$	transformační matice, reprezentující banku filtrů
$\mathcal{F}$	Fourierova transformace
$g(t, r)$	cena optimální cesty do bodu $[t, r]$
$\hat{G}_x$	odhad spektrální hustoty výkonu signálu $x[n]$
$h(t)$	impulzní odezva filtru
i	index, pozice bitu
$\mathbf{I}$	matice identity
j	imaginární jednotka
k, l, m, n	indexovací proměnné
l_{ram}	délka rámce
m	mantisa
N	počet vzorků
$o(i)$	i-tý prvek vektoru rozpoznávané sekvence
$\mathbf{o}(t)$	vektor rozpoznávané sekvence
$\mathbf{O}$	rozpoznávaná sekvence
p	operátor derivace v Laplaceově transformaci
p_{ram}	překrytí rámce
$\mathbf{q}$	výstupní vektor Fourierovy transformace

Q	komplexní číslo
Q_{Im}, Q_{Re}	imaginární a reálná složka komplexního čísla Q
r	index vektoru vzorové sekvence
$r(k)$	funkce pro výběr vektoru vzorové sekvence pro k-tý krok srovnání
R	délka vzorové sekvence, odpor rezistoru
s_{ram}	posun rámce
τ	čas jako integrační proměnná v definici konvoluce
t	čas, index vektoru rozpoznávané sekvence
$t(k)$	funkce pro výběr vektoru rozpoznávané sekvence pro k-tý krok srovnání
T	vzorkovací perioda, délka rozpoznávané sekvence
$\mathbf{T}$	transformační matice, realizující DCT
U	elektrické napětí
$v(i)$	i-tý prvek vektoru vzorové sekvence
$\mathbf{v}(r)$	vektor vzorové sekvence
$\mathbf{V}$	vzorová sekvence
$w[i]$	váha i-tého bitu
$w[k]$	váha při výpočtu DCT
$w(k)$	váha k-tého kroku srovnání sekvencí
$w[n]$	okno
W	pomocná hodnota v algoritmu FFT
x	číselná hodnota, reprezentovaná slovem; logaritmovaná hodnota
$x[n]$	diskrétní signál, řada vzorků
$x(t)$	vstupní spojitý signál
$X[n], Y[k], Z[k]$	Fourierovy transformace řad $x[n]$, $y[k]$ a $z[k]$
$X_{DCT}[k]$	kosinová transformace diskrétního signálu $x[n]$
y	výstupní hodnota při výpočtu logaritmu
$y[i]$	i-tý prvek vektoru $\mathbf{y}$
$y[k], z[k]$	řady vzorků, odvozené z $x[n]$
$\mathbf{y}$	výstupní vektor logaritmování
$\mathbf{y}^*$	výstupní vektor logaritmování, pokud při FFT dojde k dělení mezivýsledku dvěma
y_{min}	práh pro oříznutí výstupního vektoru logaritmování
$y_o[i]$	i-tý prvek vektoru $\mathbf{y}_o$
$\mathbf{y}_o$	oříznutý výstupní vektor logaritmování
$\hat{Z}_x$	odhad „poměrné přenesené energie“ signálu $x[n]$ (vztah (2.5))
Z	impedance
$\mathbb{Z}$	množina celých čísel

Seznam příloh

Součástí diplomové práce je kompaktní disk nebo `zip` archiv s tímto obsahem:

1. Adresář `CodeWarrior` obsahuje softwarovou část hlasového modulu, tedy kompletní projekt pro vývojové prostředí Freescale CodeWarrior verze 5.9.0. Zdrojové soubory jsou uloženy v podadresáři `CODE`. Většina souborů byla vygenerována automaticky, funkční části kódu jsou obsaženy především v souborech `hlasovy_modul.c` a `Events.c`.

2. Adresář `Eagle` obsahuje schémata zapojení a návrhy desek plošných spojů hlasového modulu a programátoru. Vše je vytvořeno v programu Eagle verze 5.3.0 (Light Edition). Jsou zde přiloženy i exporty schémat do formátu `pdf`.

3. Adresář `TeX` obsahuje veškeré zdrojové soubory práce pro sazbu pomocí systému LaTeX 2_ε. Podadresář `obr` pak obsahuje obrázky použité v práci, konkrétně

   (a) rastrové obrázky ve formátu `png`

   (b) vektorové obrázky ve formátu `pdf`. Většina vektorových obrázků byla vytvořena v programu AutoCAD verze 2002 Cz (zdrojový výkres `_obrázky.dwg` je taktéž přiložen) a exportována programem PDFCreator verze 0.9.7.

4. Adresář `Matlab` obsahuje

   (a) skripty s pomocnými výpočty, vytvořené v MATLABu verze R2007a

   (b) model digitálního filtru, vytvořený v Simulinku verze 6.6 a zmiňovaný v kapitole 4.3.1

   (c) ukázkový zvukový soubor `rec.wav`, na který odkazují kapitoly 2.3.1 a 2.3.2

   (d) podadresář `Test`, který obsahuje skripty pro ověření funkčnosti algoritmu, vzory hlasových příkazů a příkazy, určené k rozpoznání. Bližší popis je uveden v kapitole 2.7.

Kompaktní disk navíc obsahuje elektronickou verzi diplomové práce ve formátu `pdf`.