VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

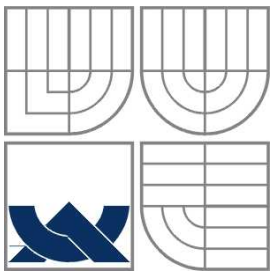
ALGORITMY ROZPOZNÁVÁNÍ ŘEČI NA FPGA/DSP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

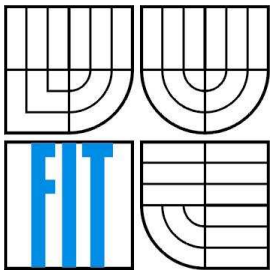
AUTOR PRÁCE
AUTHOR

Bc. OLDŘICH URBIŠ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ALGORITMY ROZPOZNÁVÁNÍ ŘEČI V FPGA/DSP

ALGORITHMS FOR SPEECH RECOGNITION IN FPGA/DSP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Oldřich Urbiš
AUTHOR

VEDOUCÍ PRÁCE — Ing. Igor Szöke
SUPERVISOR

BRNO 2008

Abstrakt

Tato diplomová práce se zabývá návrhem algoritmů pro rozpoznání řeči s ohledem na výběr cílové technologie, kterou je platforma využívající technologie signálových procesorů a programovatelných hradlových polí. Algoritmy pro rozpoznávání řeči zahrnují, extrakci příznaků v podobě Melfrekvenčních cepstrálních koeficientů, skryté Markovovy modely a jejich vyhodnocení pomocí Viterbiho algoritmu.

Klíčová slova

rozpoznávání řeči, DSP, FPGA, skryté Markovovy modely, signál, MFCC, Viterbiho algoritmus

Abstract

This master's thesis deals with design of speech recognition algorithms with consideration of target technology, which is platform combinating digital signal processing and field programmable gate array. Algorithms for speech recognition includes: feature extraction of Melfrequency cepstral coefficients, hidden Markov models and their evaluation by Viterbi algorithm.

Keywords

speech recognition, DSP, FPGA, hidden Markov models, signal, MFCC, Viterbi algorithm

Citace

Urbiš Oldřich: Algoritmy rozpoznávání řeči v FPGA/DSP. Brno, 2008, diplomová práce, FIT VUT v Brně.

Algoritmy rozpoznávání řeči v FPGA/DSP

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Igora Szökeho.
Další informace mi poskytli doc. Pavel Zemčík a Ing. Martin Žádník.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Oldřich Urbiš
19. 5. 2008

Obsah

1 Úvod

Už od počátku rozvoje lidské společnosti je potřeba mezilidské komunikace jednou ze základních potřeb. S rozvojem moderních výpočetních prostředků a zvyšováním jejich výkonnosti vznikala nutnost efektivní komunikace mezi člověkem a strojem. V počátcích mohl člověk komunikovat s počítači pouze v řeči počítačů, kterou je binární soustava, tento jazyk však není příliš přívětivým jazykem.

Aby byl nastartovaný vývoj dále udržitelný, bylo nutné vytvořit prostředky umožňující efektivnější komunikaci mezi člověkem a jím vytvořenými výpočetními prostředky. Proto se začaly v komunikaci uplatňovat dálnopisné stroje a člověk mohl začít s počítači komunikovat pomocí, pro něj přijatelnějším způsobem, písmenné reprezentace jednoduchých příkazů, avšak tento způsob, který je dnes v podstatě využíván pouze u serverových systémů a jejich operačních systémů, je poněkud těžkopádný v případě, že počítač využívá člověk, který není v oblasti výpočetní techniky příliš sběhlý. Dalším nevyhnutelným krokem byl vývoj grafických uživatelských rozhraní, které značnou měrou přispěly k zefektivnění komunikace mezi člověkem a stroji. Ruku v ruce s vývojem sofistikovanějších uživatelských rozhraní probíhal i vývoj periferních zařízení umožňující efektivnější komunikaci, jako byly myši, trackbally, tablety, atd..

Nejpřirozenějším prostředkem komunikace člověka s okolím je řeč, proto bylo logickým vyústěním vývoje počítačových rozhraní, vznesení požadavku na možnost komunikace s počítači přirozeným jazykem. Takovýto typ komunikace by značně přispěl ke zefektivnění práce a zvýšení uživatelského komfortu při komunikaci s počítači, ale nejen s nimi. Potenciálních možností využití rozpoznávání řeči je nepřeberné množství, od rozpoznání mluveného slova a jeho převod na text, až po hlasové ovládání domácích spotřebičů, a v neposlední řadě také jako pomoc pro tělesně postižené, kteří jsou odkázání na pomoc ostatních lidí, pro než by například hlasem ovládaná domácnost znamenala značné zvýšení míry jejich soběstačnosti.

V současné době se rozpoznávání řeči široce používá např. u mobilních telefonů, které umožňují vytáčení telefonních čísel, příp. ovládání telefonu, hlasovými pokyny. Tento typ rozpoznání hlasu využívá porovnání s předem nahranými vzory, se kterými se hlas pouze porovnává a vyhodnocuje se míra shody, což je způsob výpočetně nenáročný avšak také nevhodný pro jiné aplikace.

V případě výše zmíněného případu mobilních telefonů by bylo velice přínosné, kdyby mobilní telefon uměl převádět mluvené slovo na text, uživatel by mohl např. psát textovou zprávu aniž by musel vytahovat mobilní přístroj z kapsy nebo by mohl psát SMS při řízení automobilu, díky čemuž by došlo k velkému snížení rizika nehody vlivem nepozornosti řidiče.

Bohužel mobilní telefony dosud nedisponují natolik dostatečným výpočetním výkonem aby byly schopny převádět mluvenou řeč na text.

Výše uvedený případ byl pouze jedním ze zástupců potenciální množiny využití rozpoznávání řeči. V praxi by se dala najít spousta jiných případů.

Problémem rozpoznávání řeči je vysoká výpočetní náročnost používaných algoritmů, z čehož plynou značná omezení, zejména pro vestavěné aplikace, jako jsou mobilní telefony, domácí spotřebiče atd.. Proto jsou algoritmy rozpoznávání řeči vhodnými kandidáty na použití specializovaných hardwarových prostředků, které by umožnily jejich efektivní použití i pro zařízení, jejichž výkon je značně nižší než výkon dnešních počítačů. Dalším argumentem pro použití specializovaných hardwarových prostředků je zejména jejich nízká spotřeba, která je důležitá zejména u baterií napájených zařízení.

Jako vhodné prostředky, které jsou svým výkonem a spotřebou vhodné pro rozpoznávání řeči, se jeví použití digitální signálových procesorů (DSP) a programovatelných hradlových polí (FPGA).

Tato práce má za cíl zhodnotit možností současných DSP procesorů v kombinaci s využitím FPGA a provést implementaci vybraných algoritmů zpracování řeči do zvolených výpočetních prostředků.

Práce navazuje na dříve vypracované díla, která se zabývala implementací jednoduchého rozpoznávače řeči ve vývojovém kitu firmy Texas Instruments, které vypracoval Tomáš Král viz. [13] a [16].

V kapitole 2. bude podrobně probrána problematika zpracování signálů, spolu s jejich matematickým pozadím, které je nezbytné k pochopení problematiky. Na těchto základech bude vystavěna problematika rozpoznávání řeči z teoretického hlediska, bude také stručně probrána historie rozpoznávání a syntézy řeči, a také fyzikální podstata tvorby řečového signálu.

V 3. kapitole zabývající se zpracováním řeči bude také lehce vstoupeno do oboru umělé inteligence, zejména pak části zabývající se neuronovými sítěmi, které jsou jedním s prostředků pro rozpoznávání řeči a skrytých Markovových modelů, jenž budou použity pro implementaci.

Aby bylo možné provést implementaci vhodných algoritmů do výše uvedených hardwarových prostředků, je nutné co možná nejlépe znát vlastnosti a architektury použitých hardwarových a softwarových prostředků, proto je jejich popisu věnována kapitola 4.

Návrh implementace je proveden v kapitole 5.

V kapitole 6. je podrobně popsána samotná implementace rozpoznávače a v závěrečné 7. kapitole je provedeno zhodnocení práce a jejich výsledků.

2 Teorie zpracování signálů

V následující kapitole bude probrána problematika zpracování signálů ať už spojitých nebo diskrétních. Budou zde zmíněny základní pojmy, které jsou nutné pro popis dané problematiky, zejména z oblasti diskrétních signálů, neboť tyto jsou klíčové při zpracování signálu pomocí číslicových výpočetní prostředků. Zpracování diskrétních signálů zahrnuje také nutnost jejich převodu ze spojité domény do domény diskrétní, z čehož plynou níže popsané problémy, zejména problémy se vzorkováním a vznikem aliasingu.

Nedílnou součástí zpracování signálů je jejich matematický popis, který bude také v této kapitole uveden. Klíčovou roli v analýze a zpracování signálů, ať už audio nebo video signálů , hraje Fourierova řada a její transformace, to pro případ signálů spojitých. V případě signálů diskrétních je důležitým prostředkem pro jejich popis diskrétní Fourierova řada a její transformace, příp. inverzní diskrétní Fourierova transformace. Matematický základ těchto transformací bude také uveden.

.

2.1 Spojité signály

Protože zpracování řeči je založeno na pořizování a zpracování akustického signálu je nutné definovat pojem signálu a s tím spojené pojmy, jako jsou konvoluce, energie a výkon signálu atd..

2.1.1 Definice signálu

Signál můžeme definovat jako jakoukoliv energii, která je použita k přenosu nějakého typu informace. Signál může být matematicky vyjádřen jako funkce s jednou nebo několika nezávislými proměnnými a jednou proměnnou závislou. Spojitý signál je definován v jakémkoliv časovém okamžiku.

Pro potřeby této práce, budeme uvažovat akustický signál, který používá k přenosu informace změny tlaku vzduchu v akustickém prostředí v závislosti na čase.

Aby bylo možno akustický signál dále zpracovávat elektronickými zařízeními, ať už analogovými nebo číslicovými, je nutné akustický signál transformovat na signál elektrický, jež používá k přenosu informace změny velikosti elektrického proudu.

Důležitým signálem při zpracování signálu je signál harmonický, který můžeme vyjádřit vztahem

$$f(t) = A \cdot \sin(\omega t + \varphi_0) \tag{1}$$

Rovnice harmonického signálu.

.

2.1.2 Energie a výkon signálu

Jak již bylo zmíněno v předchozí podkapitole, signálem rozumíme přenos informace pomocí měnící se úrovně energie, proto jsou charakteristiky velikosti energie a průměrného výkonu velice důležité při zpracování signálů. Energie signálu je definována určitým integrálem v intervalu $[t_1, t_2]$, a můžeme ji vyjádřit vztahem

$$E = \int_{t_1}^{t_2} \left| x(t) \right|^2 dt \tag{2}$$

Energie.

.

Průměrný výkon signálu v intervalu $[t_1, t_2]$ je pak definován vztahem

$$P = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \left| x(t) \right| dt \tag{3}$$

Průměrný výkon.

kde:

E……energie signálu na daném intervalu

P……průměrný výkon signálu na daném intervalu

x(t)….signál

.

2.1.3 Konvoluce

Dalším z řady pojmů týkajících se zpracování signálu je konvoluce. Konvoluce se široce využívá u lineárních časově invariantních systémů, které mají odezvu na základě funkce impulsní odezvy. Využití konvoluce je markantní zejména u signálových filtrů.

Konvoluci můžeme definovat jako matematický operátor, který za své argumenty přijímá dvě funkce *f* a *g* a jehož výsledkem je třetí funkce, která v podstatě reprezentuje velikost překrytí mezi fcí. *f* a převrácenou a posunutou fcí. *g*.

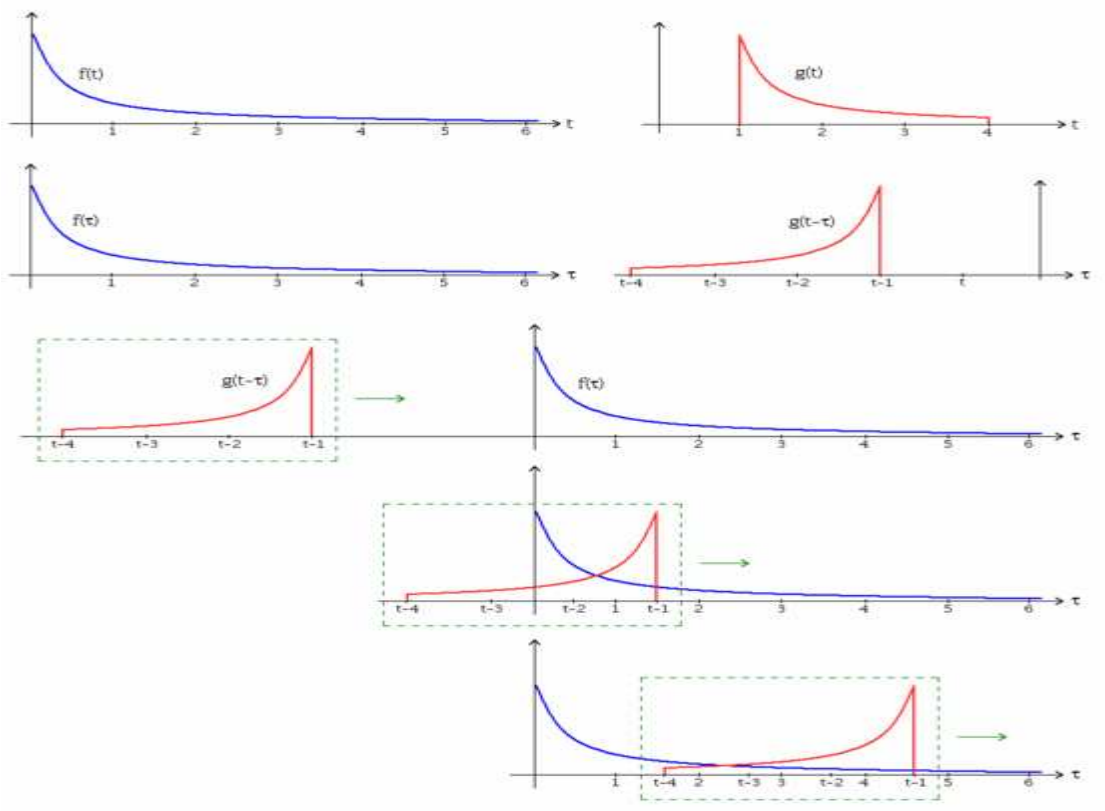
Obvykle je jedna z funkcí používána jako pevná impulsní odezva a je nazývána konvolučním jádrem.

Matematické vyjádření konvoluce pro spojité signály je možno vyjádřit následujícím matematickým vztahem.

$$(f * g)(t) = \int_a^b f(\tau) g(t - \tau) d\tau \,. \tag{4}$$

Konvoluce.

Na následujícím obrázku je grafické znázornění konvoluce dvou spojitých signálů.

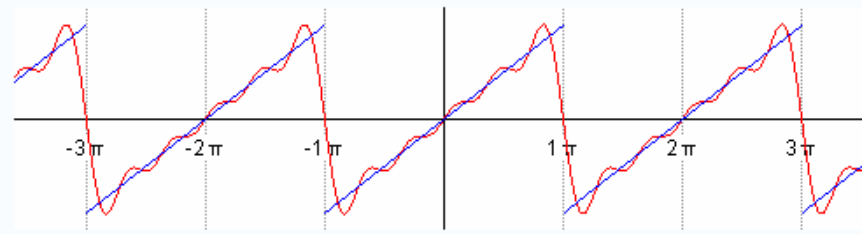


Obrázek 1. Konvoluce spojitých signálů[7].

2.1.4 Fourierova řada

Jedním ze základních matematických prostředků používaných pro analýzu spojitých signálů je Fourierova analýza, kterou zavedl jako první francouzský matematik Jean Baptiste Fourier(1786-1830)[4].

Fourierova řada vychází z Fourierova teorému viz.[4], který říká, že libovolnou křivku, která má libovolné vlastnosti a byla získána jakýmkoliv způsobem, je možné reprodukovat složením dostatečného počtu harmonických křivek, jak je ukázáno na následujícím obrázku.



Obrázek 2. Aproximace funkce Fourierovou řadou. Převzato z [5].

Definici Fourierovy řady můžeme zapsat například takto: Z [8]. Nechť funkce *f(x)* je integrovatelná v intervalu $\langle c, c+2\pi \rangle$. Pak trigonometrickou řadu

$$\frac{a_k}{2} + \sum_{k=1}^{\infty} \left(a_k \cos kx + b_k \sin kx \right), \tag{5}$$

Rovnice Fourierovy řady[8].

kde koeficienty a_k, b_k jsou vyjádřeny revnicemi 6, nazýváme *Fourierovou řadou* funkce *f(x)* v intervalu $\langle c, c + 2\pi \rangle$. Podrobněji viz. [8].

$$\begin{aligned} a_0 &= \frac{1}{\pi} \int_c^{c+2\pi} f(x) dx \\ a_k &= \frac{1}{\pi} \int_c^{c+2\pi} f(x) \cos kx dx \\ b_k &= \frac{1}{\pi} \int_c^{c+2\pi} f(x) \sin kx dx \\ pro\ k &= 1,2,...n \end{aligned} \tag{6}$$

Koeficienty Fourierovy řady[8].

2.1.5 Fourierova transformace

Fourierova řada umožňuje popisovat pouze signály jež jsou periodické, avšak pro praktické použití je nutné mít prostředky pro popis signálů, jež jsou aperiodické, ale tyto chceme také popisovat pomocí součtu harmonických složek. Funkce $S(i\omega)$, vyjádřená v rovnicí 7, se nazývá Fourierův obraz signálu *x(t).*

$$S(i\omega) = \int_{-\infty}^{+\infty} s(t) e^{-i\omega t} dt \tag{7}$$

Fourierův obraz fce. *s(t)* [9].

Funkci *s(t)* dostaneme z fce. *S(iω)* pomocí inverzní Fourierově transformaci uvedené v rovnici 8.

$$s(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\omega) e^{i\omega t} d\omega \tag{8}$$

Inverzní Fourierova transformace[9].

S ohledem na rozsah nebylo uvedeno odvození výše uvedených vztahů, toto je možné získat z [6], [8] a [9].

kde:

i……..imaginární složka

ω…….úhlová frekvence

s(t)….funkce popisující signál

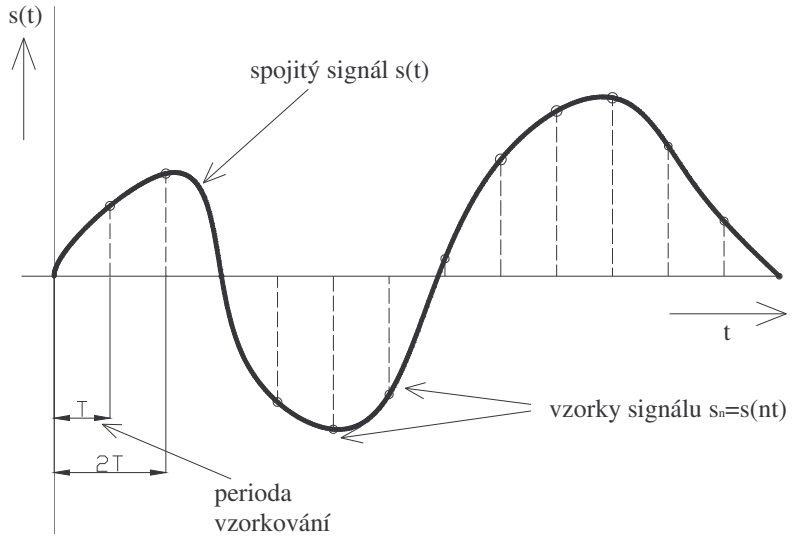
2.2 Diskrétní signály

Naprostá většina současných prostředků pro zpracování signálů, ať už akustických, elektrických nebo jakýchkoliv jiných, je založena na číslicovém zpracování, z tohoto důvodu je nutné aby zpracovávaný signál byl v jeho diskrétní reprezentaci. A tedy jakýkoliv spojitý signál musí být nejprve diskretizován, což se provádí pomocí vzorkování signálů, které je popsáno v podkapitole 2.2.1. Takovýto signál je pak možné zpracovávat na číslicových výpočetních prostředcích pomocí níže popsaných nástrojů, jako jsou diskrétní Fourierova řada, podkapitola 2.2.2, a Fourierova transformace s diskrétním časem, podkapitola 2.2.3. Oba tyto nástroje vycházejí z výše definovaných Fourierových řad a Fourierových transformací pro signály se spojitým časem.

2.2.1 Vzorkování a rekonstrukce

Mluvená řeč, která přenáší informaci, je přenášena změnou akustického tlaku v určitém místě prostoru, aby ji bylo možno zpracovávat, převádí se změny akustického tlaku pomocí nějakého zařízení, obvykle mikrofonu, na elektrický signál. Zmiňovaný signál je však spojitý v čase a tudíž je nutné jej pomocí vzorkování a kvantizace převést na signál s diskrétním časem. Takový signál je pak reprezentován posloupností číselných hodnot z konečného oboru hodnot. Proces se nazývá pulsně kódová modulace nebo též digitalizace[2].

Prvním krokem převodu spojitého signálu na signál diskrétní je vzorkování. Vzorkování je transformace signálu $s(t)$ spojitého v čase, na posloupnost vzorků $s_n=s(nT)$ diskrétních v čase. Toto vzorkování probíhá v časových okamžicích $t_n=nT$, kde T je perioda vzorkování a $n=0, \ldots, \infty$ podrobněji viz. [2]. Schematický nákres vzorkování spojitého signálu je znázorněn na obrázku 3.

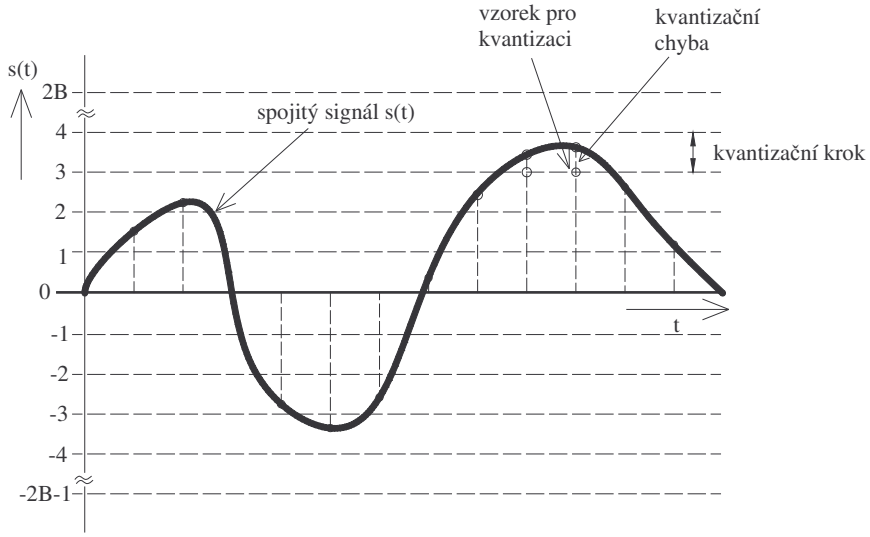


Obrázek 3. Schéma vzorkování spojitého signálu.

Vzorkování musí být prováděno na frekvenci, která je dvakrát větší než nejvyšší frekvence, která se ve spojitém signálu vyskytuje. Omezující podmínka vyplývá z Nysquitova vzorkovacího teorému. Pro vzorkovací frekvenci tedy musí platit, že $F_v=2 \, . \, F_{max}$ viz. [2].

Dalším krokem při převodu spojitého signálu na signál diskrétní je kvantizace a kódování. Kvantizace s následným kódováním je proces při kterém se aproximují analogové hodnoty signálu jednou hodnotou z konečného počtu hodnot. Tento převod je většinou prováděn analogově-digitálním převodníkem, který má na vstupu hodnotu napětí a na výstup generuje příslušnou zakódovanou hodnotu. Protože je kódování prováděno do konečného oboru hodnot, tak při převodu uvažujeme kvantizační krok, což je minimální rozdíl napětí, který je nutné ke změně hodnoty o jednu úroveň. Dále uvažujeme rozsah úrovní kvantování, který udává kolika konečnými hodnotami můžeme popsat úrovně signálu. Pro praktické aplikace se používají převodníky s bitovou šířkou 16bitů. Na rozlišení převodníku také závisí jak velký dynamický rozsah můžeme daným převodníkem zaznamenat.

Při kvantizaci vzniká chyba, tzv. kvantizační chyba, které vzniká zaokrouhlením hodnot na nejbližší celou hodnotu. Téma kvantizace a kódování je podrobně vysvětleno ve [2], kde jsou zmíněny i další možnosti kódování jiné než PCM. Na obrázku 4. je zobrazeno schéma kvantizace spojitého signálu.



Obrázek 4. Schéma kvantizace.

2.2.2 Diskrétní Fourierova řada

Diskrétní Fourierova řada je založena na výše popsané Fourierově řadě pro spojité signály, proto se ji nebudeme podrobněji zabývat a pouze zmíníme její matematický vzorec.

Dle [10] diskrétní Fourierova řada přiřazuje posloupnosti *{s(n)}* s periodou *N* obraz *{S(k)}*, periodickou posloupnost s periodou *N*:

$$S(k) = \sum_{n=0}^{N-1} s(n) \cdot \exp\left(- j \cdot \frac{2\pi}{N} \cdot kn \right) \tag{9}$$

Diskrétní fourierova řada[10].

Zpětná diskrétní Fourierova řada přiřazuje periodické posloupnosti *{S(k)}* původní posloupnost *{s(n)}*.

$$s(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} S(k) \cdot \exp\left(j \cdot \frac{2\pi}{N} \cdot kn \right) \tag{10}$$

Zpětná diskrétní Fourierova řada[10].

2.2.3 Fourierova transformace s diskrétním časem

Fourierova transformace s diskrétním časem slouží ke spektrální reprezentaci aperiodických diskrétních signálů posloupnosti konečné délky. Pro výpočet obrazu diskrétní Fourierovy transformace posloupnosti *{s(n)}* platí

$$S(k) = \sum_{n=0}^{N-1} s(n) \cdot \exp\left(- j \cdot \frac{2\pi}{N} \cdot kn \right), \tag{11}$$

kde $n=0,1,2,\ldots,N-1$.

Diskrétní Fourierova transformace[10].

Výše uvedený vztah vyjadřuje předpis pro výpočet diskrétního aperiodického signálu z N vzorků signálu. Inverzní diskrétní Fourierova transformace, pak vypadá následovně:

$$s(n) = \sum_{n=0}^{N-1} S(k) \cdot \exp\left(- j \cdot \frac{2\pi}{N} \cdot kn \right), \tag{12}$$

kde $k=0,1,2,\ldots,N-1$.

Inverzní diskrétní Fourierova transformace[10].

Fourierova transformace je vysoce výpočetně náročná, a proto vznikla rychlá Fourierova transformace, která značně snižuje výpočetní složitost celého výpočtu, detailnější popis algoritmu rychlé Fourierovy transformace, dále jen FFT z anglického Fast Fourier Transform, bude popsán až při samotné implementaci.

3 Teorie zpracování řeči

Cílem této práce je implementovat jednoduchý rozpoznávač slov ve specializovaných hardwarových prostředcích, a je tedy vhodné seznámit se s možnými přístupu k této problematice.

V následující kapitole budou jednotlivé možné přístupy diskutovány a bude popsán a vysvětlen jejich princip. Abychom mohli vůbec aplikovat nějaké přístupy k rozpoznávání řeči je také vhodné znát podstatu vzniku řečových signálů z pohledu stavby hlasového ústrojí. V této kapitole bude uvedeno schéma a popis hlasového aparátu včetně lehkého nástinu jeho funkce při generování řečového signálu.

Na začátek kapitoly je zařazen odstavec pojednávající o historii problematiky rozpoznávání i syntézy řeči, který bude vhodným doplňkem k přesnějšímu zasazení problematiky do širšího kontextu uvedeného v úvodu.

Nejdůležitější částí této kapitoly bude popis postupu rozpoznání řeči od jeho navzorkování až po jeho finální rozpoznání na základě jednoho z níže diskutovaných postupů. V praktických aplikacích se používají přístupy založené na akusticko-fonetickém přístupu, pattern recognition přístup neboli přístup založený na rozpoznání vzorů v daném akustickém signálu a nakonec přístup založený na algoritmech umělé inteligence, jako jsou např. neuronové sítě nebo skryté Markovovy modely. Aby bylo vůbec možné výše uvedené přístupy aplikovat, je třeba zmenšit objem informace, který nám poskytuje audiosignál, postup při kterém se omezuje množství vesměs nepotřebné informace se nazývá Feature extraction - extrakce příznaků, kdy se z daného signálu vyberou důležité charakteristické rysy, na základě kterých je následně možné provádět samotné rozpoznávání. Uvedené postupy budou popsány v jedné z následujících podkapitol.

3.1 Obecné pozadí zpracování řeči

Vývoj metod pro rozpoznávání řeči zahrnuje spousty vědních oborů, od biologie až po informatiku. Proto, aby vývoj metod byl efektivní, je nutné znát pozadí z většiny oborů do nichž rozpoznávání řeči zasahuje, a je také důležité mít znalost historického vývoje, díky kterým se můžeme vyhnout případným chybám, které se staly v minulosti, příp. znovu použít myšlenky jenž ve své době nemohly být realizovány, ať už z důvodu nízké technologické nebo vědomostní úrovně.

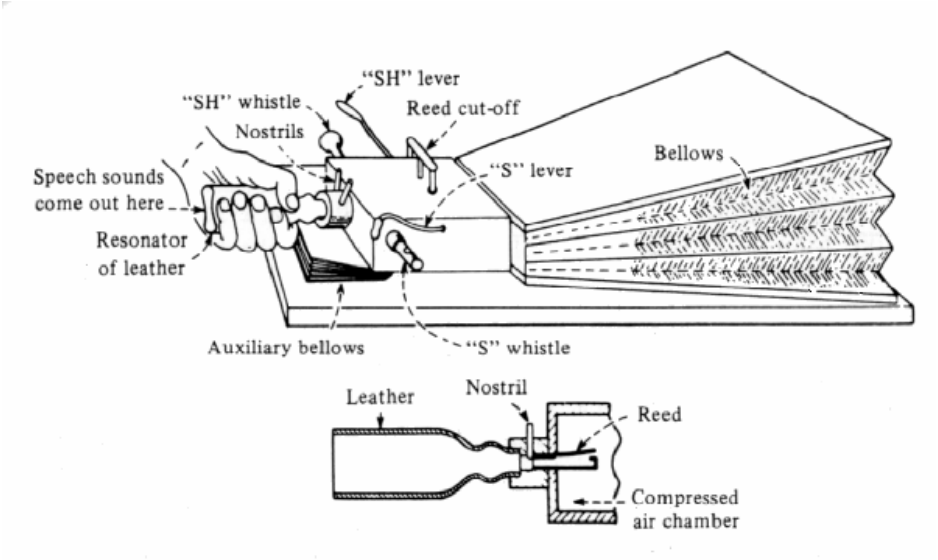
V následujících kapitolách bude probrán historický vývoj od počátku vývoje syntézy řeči, která bude zmíněna jen okrajově, až po současný, příp. nedávný vývoj. Podkapitola 3.1.1 vychází z knihy [1], kde je vývoj podrobně probrán. V podkapitole 3.1.2 bude diskutována problematika vzniku řeči z pohledu biologické stavby řečového ústrojí, tato podkapitola vychází z poznatků uvedených v [2].

3.1.1 Historie rozpoznávání řeči

3.1.1.1 Historie syntézy řeči

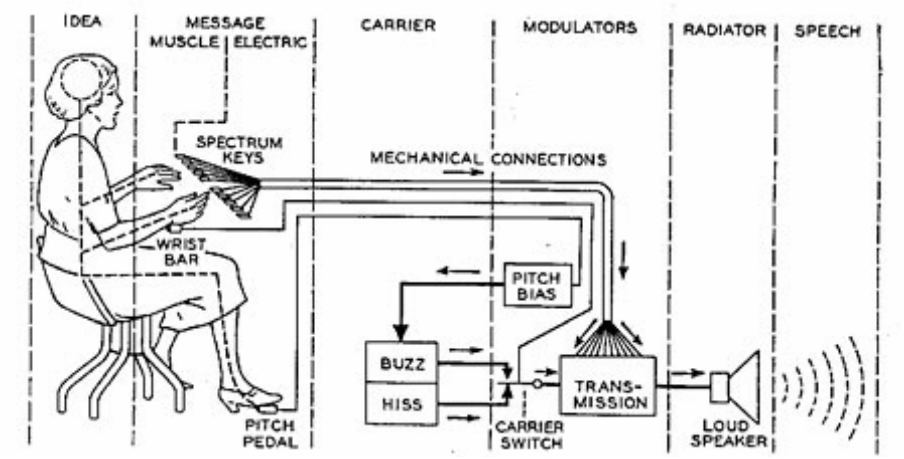
Samotná historie rozpoznávání řeči začala poznáním, že je možné řeč namodelovat, s tímto tvrzením přišel von Kempelen, který ho dokázal tím, že okolo roku 1780 sestrojil mechanický stroj, který uměl „mluvit“.

Na základě von Kempelenova stroje sestavil Wheatstone vylepšený přístroj, který byl vyroben z kůže a na němž mohl operátor simulovat chování řečového ústrojí při mluvení. Schéma Wheatstoneova stroje je uvedeno na obrázku 5.



Obrázek 5. Wheatstoneův "mluvící stroj". Převzato z [1].

Moderní způsoby syntézy řeči ve skutečnosti začaly vznikat až s vynálezem zařízení vocoder (voice-coder) a Voder (voice-operated demonstrator)[1]. Tato zařízení zkonstruoval americký vynálezce Homer Dudley. Voder měl klaviaturu podobnou s pianem a pomocí jeho kláves bylo možno modelovat hlasové ústrojí a tím simulovat řeč. Celé zařízení kladlo velké nároky na operátora, a proto bylo problém najít vhodné kandidáty, kteří by jej byli schopni zvládnout. Na obrázku 6. je schématický nákres Vocoderu.



Obrázek 6. Vocoder. Převzato z [11].

3.1.1.2 Historie rozpoznávání řeči

Počátek historie rozpoznávání řeči můžeme dotovat do roku 1920, kdy byl na svět uveden první komerční výrobek, který využíval poznatků z oblasti zpracování řeči. Byla jím hračka psa nazvaná Radio Rex, která reagovala na slovo Rex, po kterém pes vyskočil z boudy. Celý systém byl založen na tom, že slovo Rex obsahuje frekvenci 500Hz s dostatečnou energií. Na této frekvenci pak rezonoval můstek, který následně přerušil elektrický obvod a pružina, která byla držena elektromagnetem, vytlačila psa ven z boudy.

Hračka samozřejmě reagovala i na jiná slova, která měla podobnou charakteristiku, ale to nic nezměnilo na jejím úspěchu, díky kterému došlo k dalšímu pokroku v rozpoznávání řeči.

Dalším, už v mnohém pokrokovějším systémem, byl číselný rozpoznávač vyvinutý v Bell Labs. Rozpoznávač uměl na základě vysloveného čísla správně určit, o které číslo se jedná. Celý systém bylo nutné naučit na jednoho určitého mluvčího, avšak chyba při rozpoznání pak byla pouhá 2 procenta.

Celý systém byl založen na prostém měření spektrální energie v čase ve dvou širokých pásmech, hrubě aproximující první dva formanty. Detailní způsob zpracování řečí tímto zařízením je popsán v [1], včetně schématu zařízení.

V roce 1958 vytvořil Dudley klasifikátor, který nepřetržitě vyhodnocoval spektrum, namísto aproximace formantů. Tento nový přístup byl následně často využíván.

V roce 1959 přidal Danes, působící na Collage of London, k akustické informaci také informaci o pravděpodobnosti gramatiky. Jinými slovy řekl, že pravděpodobnost výskytu lingvistické jednotky závisí také na předešlém výskytu jiné lingvistické jednotky, tudíž pravděpodobnost výskytu slova není založena jenom na akustickém vstupu.

Další vývoj metod pro rozpoznávání řeči pokračoval širokým vývojem v šedesátých letech, kdy v roce 1964 byly Martinem poprvé použity neuronové sítě pro klasifikaci fonémů. Řečové klasifikátory se dále zlepšovaly v úspěšnosti rozpoznávání už i pro několik různých mluvčích.

Během šedesátých letech došlo k několika významným zlepšením doposud používaných metod pro kódování řeči, jako jsou použití rychlé Fourierovy transformace, cepstrální analýza, Linear predictive coding (LPC), a pro rozpoznání vzorů jsou to deterministický Dynamic time warp (DTW) a statistický Hidden Markov model (HMM).

V letech 1971-1976 byl spuštěn ARPA projekt, který měl za cíl vytvořit rozpoznávač, jež by byl schopný automaticky rozpoznávat až 1000 slov.

V osmdesátých letech pokračoval nastartovaný vývoj, který rozšiřoval a vylepšoval stávající metody rozpoznávání řeči, zejména s ohledem na zvyšující se výkon tehdejších výpočetních prostředků. V tehdejších rozpoznávačích se používaly metody založené na HMM, neuronových sítí, které se zase začaly prosazovat po tom co v roce 1969 došlo díky knize Perceptrony od Minsky a Paperta k ochladnutí zájmu o neuronové sítě, a rozpoznávačích založených na znalostní bázi.

V roce 1984 byl spuštěn druhý projekt ARPA, který měl za úkol vyvinout systém, díky kterému by bylo možné klást otázky mluvenou řečí přímo nad databází námořnictva. Tento projekt se vyvinul tak, že v roce 1998 už byl schopný rozpoznávat slova ze slovníku o 60 000 slovech, systém se nemusel „naučit“ nového mluvčího a chybovost rozpoznání slov byla pod 10%.

Takto nastartovaný vývoj pokračuje dále do dnešní doby.

3.1.2 Funkce hlasového ústrojí

Řeč vzniká v hlasovém ústrojí, ve kterém se proud vzduchu moduluje na samotnou řeč. Za jeden ze základních kamenů hlasového ústrojí můžeme považovat plíce, jež jsou základním zdrojem energie proudícího vzduchu. Vzduch, který je vytlačován pod tlakem z plic přes průdušnici, prochází další nezbytnou součástí celého ústrojí, kterou jsou hlasivky. Hlasivky tvoří v podstatě „hráz“, jež brání vzduchu v jeho proudění. Ve svém klidovém stavu jsou plně uzavřeny a až dojde k vytlačení vzduchu z plic a nárůstu tlaku, tak se hlasivky rozkmitají a střídavým zavíráním a otevíráním dochází k „dávkování“ vzduchu a tím k utváření tónu.

Tón vznikající v hlasivkách se nazývá základní a představuje nosný zvuk řeči, jeho frekvence bývá většinou v rozsahu 60-400Hz [2] v závislosti na jedinci se může i výrazně lišit. Základní tón se může v čase mírně měnit, a to jak ve frekvenci (jitter), tak i v amplitudě (shimmer).

Hlasivky jako takové k řeči nestačí, základní tón je třeba modulovat za přispění nadhrtanové dutiny, jež je pasivním prvkem, a artikulačními orgány, jež jsou prvkem aktivním. Artikulační orgány, jako jsou jazyk, měkké patro, rty, svým pohybem mění velikost nadhrtanových dutin, čímž dochází k modulaci základního tónu a vzniku řeči. Problematika tvorby řeči je detailněji vysvětlena v [1] a [2], ze kterých čerpala tato podkapitola.

3.2 Možné přístupy

Při rozpoznávání řeči je možné využít několik přístupů použitelných pro rozpoznávání. Jednotlivé přístupy budou popsány v následující podkapitolách.

3.2.1 Akusticko fonetický přístup

Dle [13] je akusticko-fonetický přístup založený na rozpoznávání jednotlivých foném a jejich vlastností, specifických pro jednotlivé jazyky. Vlastnosti fonémů se mohou velice lišit, zejména s ohledem na řečníka, ale i v závislosti na sousedních fonémech. To znamená, že jeden foném může mít různé vlastnosti jenom proto, že je ve spojení s různými fonémy. Pro praktické aplikace se využívá uskupení tří za sebou jdoucích fonémů, které se označují jako trifóny. Trifóny vznikají jako všechny možné kombinace fonémů. Je zřejmé, že takovýchto možných kombinací je značné množství, i když velká část z nich je neexistujících, avšak i tak je rozpoznávání založené na rozpoznávání trifónů značně výpočetně náročné. Nicméně pro svou vysokou úspěšnost při rozpoznávání je široce využíváno, a to zejména v kontextu rozpoznávání spojité promluvy s velkým slovníkem, kde není možné natrénovat všechna slova ze slovníku neboť by takovéto trénování bylo praktický nerealizovatelné. Dalším aspektem, který by v případě použití takovéhoto způsobu rozpoznávání hrál velmi významnou roli, je výpočetní náročnost celého procesu, protože za velký slovník je považován takový, který obsahuje více jak cca 50000 slov, což je relativně nízká hodnota. Pro srovnání například český jazyk obsahuje až několik set tisíc slov a všechna tato slova by bylo nutné v reálném čase ohodnotit a určit nejlepší shodu.

3.2.2 Přístup založený na rozpoznání vzorů

Přístup založený na rozpoznání vzorů využívá schopnosti určitých algoritmů porovnávat mezi sebou vstupní vzor se vzory z množiny vzorů. Výsledkem je většinou pravděpodobnost shody vstupního vzoru s jednotlivými vzory z množiny vzorů. Jedním z běžně používaným algoritmů pro porovnávání vzorů je výpočet Euklidovy vzdálenosti, které nám říká jak moc se daný objekt liší od svého vzoru. Nicméně tento přístup se při rozpoznávání řeči nepoužívá, byl zde zmíněn pouze jako příklad.

Množina vzorů je vytvářena pomocí množiny trénovacích vzorů, na kterou je daný algoritmus, např. neuronové sítě a jejich učící algoritmy, natrénován. Po úspěšném natrénování je algoritmus připraven na samotnou klasifikaci. Výše popsané přístupy je možné kombinovat.

3.3 Postup rozpoznávání

Rozpoznávání mluvené řeči se provádí v několika krocích. Nejprve je potřeba vstupní signál převést z analogové reprezentace na reprezentaci digitální pomocí vzorkování a kvantování, které jsou

popsány v kapitolách 2.2.1 a 2.2.2. Následně se signál, už diskrétní, musí rozložit na jednotlivé rámce. Jednotlivé rámce jsou následně násobeny Hammingovým oknem, aby se minimalizoval vliv vzorků, které nejsou ve vybraném rámci. Na takto upravené okno se aplikují algoritmy pro extrakci příznaků pomocích kterých se provádí následné porovnávání.

3.3.1 Extrakce příznaků

Rozpoznávání řeči je založeno na zpracování akustického signálu, avšak tento signál nese velké množství informace, kterou nejsme schopni v reálném čase zpracovávat, z toho důvodu je nutné aby se velikost informačního toku snížila na maximální možnou úroveň přičemž zůstane zachována většina užitečné informační hodnoty. Procesem, kterým je možné tohoto dosáhnout je extrakce příznaků (anglicky „feature extraction“). Extrakce příznaků se snaží vyjádřit řečový signál omezeným množstvím hodnot..

Při extrakci příznaků se používají dva možné přístupy, jimiž jsou neparametrický popis, který je založen pouze na metodách pro zpracování signálu, jako jsou banky filtrů nebo Fourierova transformace viz. [12], a parametrický popis, který je založen na poznatcích o tvorbě řeči, a který rovněž využívá metod použitých u neparametrického popisu, ale pouze jako podpůrných prostředků.

Dle [12] parametry dělíme na skalární a vektorové. Kde skalárními rozumíme jedno číslo nebo řečový úsek a vektorovými rozumíme sadu čísel nebo řečový úsek, přičemž v případě více řečových úseků, řadíme vektorové hodnoty do matic.

Podrobný popis jednotlivých kroků, které je nutné provést, abychom ze vstupního audiosignálu obdrželi požadované příznaky, jsou popsány v následujících několika odstavcích.

3.3.1.1 Předzpracování vstupního signálu

Před samotnou extrakcí příznaků je vhodné vstupní signál předzpracovat, tak aby by byla maximálně zvýšena jeho užitečná informace a odstraněny nežádoucí vlivy, které by mohly negativně ovlivnit kvalitu rozpoznání.

Úroveň vstupního signálu, který je pořizován nějakým audio zařízením např. zvukovou kartou počítače, může mít, vlivem konstrukce a funkce daného zařízení, posunutou střední hodnotu. Při rozpoznání řeči člověkem toto nehraje významnější roli, neboť lidské ucho je takovou anomálii v signálu schopno kompletně vyeliminovat. Avšak v případě strojového rozpoznávání už může vzniknout problém. Například při výpočtu hodnoty energie signálu může tato energie být vyšší nebo nižší než je skutečná hodnota, což v důsledku bude mít vliv, pokud budeme chtít detekovat promluvu ve spojitém signálu, protože se dostaneme mimo prahové hodnoty a detektor bude chybně detekovat.

Proto je vhodné provést ustředění signálu, kdy se střední hodnota signálu posune na nulovou úroveň. Ustředění je možné provádět offline nebo online. Offline ustředění provede pouze výpočet průměrné střední hodnoty dle vzorce:

$$\bar{s} = \frac{1}{N}\sum_{n=1}^{N} s[n] \tag{13}$$

Při online ustředění se střední hodnota signálu počítá rekurzivně. A signál se ustřeďuje postupně s postupujícím časem. Tento přístup je vhodný pro souvislý signál, kdy není možné vzít celý signál a ustředit ho offline. Další výhodou online ustředění je ustředění signálu, i když dojde ke změně střední hodnoty během jeho zpracovávání.

V obou výše uvedených případech se výsledný odhad střední hodnoty signálu odečte od příslušného signálu, čímž jej ustředíme.

Protože dle [12] klesá energie řeči směrem k vyšším frekvencím je vhodné, před samotným zpracováním extrakcí příznaků, využít preemfáze, která se stará o vyrovnání kmitočtové charakteristiky řeči, a tedy o zesílení energetický slabých vysokých frekvencí.

Preemfázi je možné realizovat pomocí jednoduchého filtru 1. řádů, který můžeme vyjádřit rovnicí

$$H(z) = 1 - \kappa z^{-1}, \tag{14}$$

Rovnice filtru 1. řádu[12].

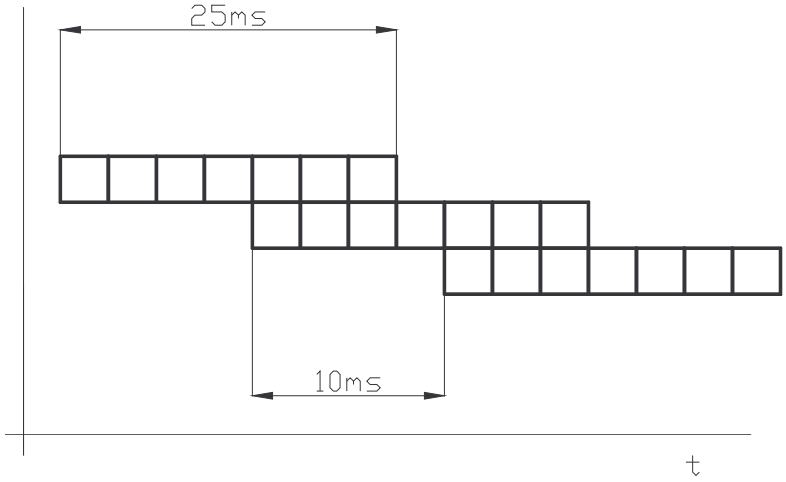
kde $\kappa \in [0.9,1]$.

3.3.1.2 Segmentace vstupního signálu na rámce

Řečový signál považuje za náhodný, avšak pro metody odhadu potřebujeme, aby signál byl stacionární, tudíž je nutné ho nějakým způsobem na stacionární signál převést. Toto nám umožní rozdělení vstupního signálu na rámce, které jsou pak následně zpracovávány.

Velikost rámce zvolíme dostatečně malou, aby se signál choval stacionárně, ale zároveň dostatečně velkou, aby nedošlo ke ztrátě podstatné informace. Velikost rámce volíme cca 20-25ms s překrytím cca 10ms. Hodnota 20-25ms je zvolena na základě setrvačnosti hlasového ústrojí.

Na každý rámec je následně aplikováno Hammingovo okno dle níže uvedeného vztahu, jehož úkolem je dle [12] utlumit signál na okrajích, protože při použití obdélníkového okna dojde v místech „vyseknutí“ rámce k zanesení nežádoucích vysokých frekvencí, které zhoršují frekvenční charakteristiku signálu.



Obrázek 7. Schéma překrytí rámců.

$$w[n] = \begin{cases} 0.54 - 0.46 \cos \frac{2\pi n}{l_{ram} - 1} & pro\ 0 \le n \le l_{ram} - 1 \\ 0 & jinde \end{cases} \qquad (15)$$

Hammingovo okno[12].

3.3.1.3 Banky filtrů

Abychom mohli ze vstupního signálu extrahovat potřebné parametry, je nutné nejprve ze zpracovávaného signálu získat jeho spektrum. To můžeme udělat Fourierovou transformací aplikovanou na aktuální rámec. Po umocnění vypočtených komplexních čísel získáme výkonové spektrum zpracovávaného rámce.

Jednou z charakteristik použitých při rozpoznávání řeči jsou cepstrální koeficienty, které mají za úkol odstranit z výkonového spektra nepotřebné informace o buzení signálu a získat pouze informace popisující modifikaci signálu, tj. funkce artikulačního traktu, protože řeč je tvořena konvolucí buzení, tj. základní hlasivkový tón, a impulsní odezvou filtru, tj. artikulačního traktu. Tyto dvě složky je nutné od sebe separovat, abychom buzení z dalšího zpracování vyloučili. K tomu můžeme s výhodou využít diskrétní Fourierovu transformaci. Vzorec pro výpočet cepstrálních koeficientů můžeme zapsat ve tvaru

$$c(n) = F^{-1}\left\{\ln\left|F\left[s(n)\right]\right|^2\right\} \qquad (16)$$

Vzorec pro výpočet cepstrálních koeficientů [16].

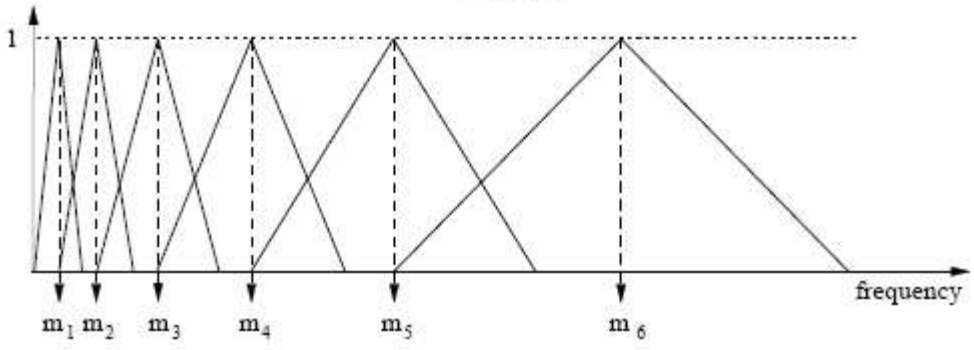
.

Cepstrální koeficienty nevystihují dostatečně přesně způsob, jakým člověk vnímá zvuk, protože předpokládají, že člověk vnímá zvuk s rozlišením, které se nemění se vzrůstající frekvencí, což není pravda. Dle [12] člověk se zvyšující se frekvencí hůře rozlišuje. Proto vznikly Mel-frekvenční

cepstrální koeficienty (MFCC), které tento nedostatek pomocí 23 nelineárních filtrů odstraňují. Vstupem pro výpočet cepster je hodnota energie, která je na výstupech těchto filtrů. Schéma rozmístění filtrů na ose je na obrázku 7. Rozmístění filtrů je možné určit na základě vztahu

$$F_{MEL} = 2959 \log_{10}\left(1 + \frac{F_{Hz}}{700}\right), \qquad (17)$$

který byl získán empirickými pokusy a vyjadřuje změnu citlivosti sluchu s ohledem na frekvenci.



Obrázek 8. Rozmístění Mel filtrů na frekvenční ose[12].

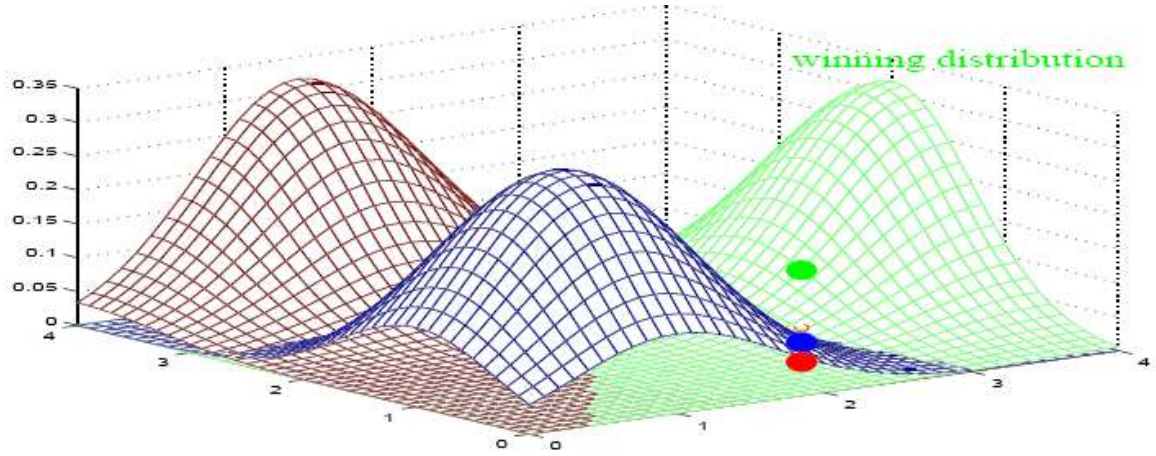
3.3.2 Rozpoznávání

3.3.2.1 Skryté Markovovy modely – Hidden Markov Models

Dle [17] jsou skryté Markovovy modely (HMM) statistickým modelem, který modeluje Markovův proces s neznámými parametry a jeho úkolem je určit skryté parametry z parametrů zjistitelných. Takto získané parametry je možné využít pro pozdější zpracování, v případě rozpoznávání řeči se jedná o rozpoznávání vzorů.

Metoda klasifikace na základě skrytých Markovových modelů vychází při své činnosti z množiny naučených vzorů, jimiž jsou slova uložená ve slovníku. Aby bylo rozpoznávání možné je potřeba vzory rozdělit vhodnou formou do jednotlivých tříd. Touto formou v případě rozpoznávání řeči a této práce bude Gaussovo rozložení pravděpodobnosti.

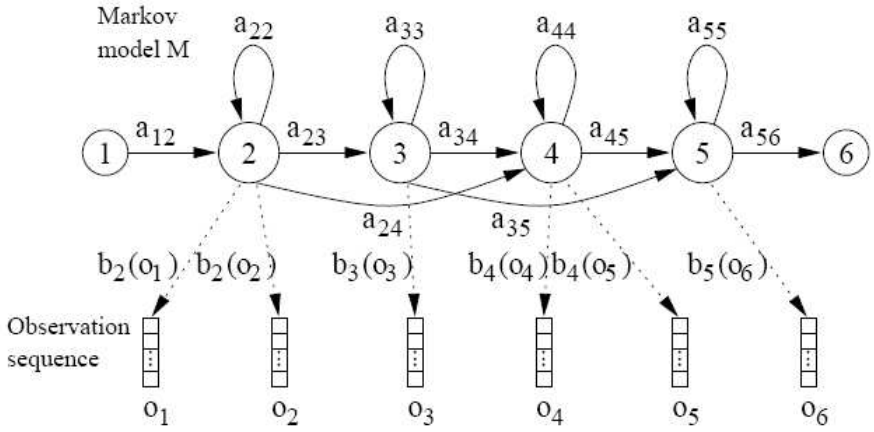
Grafické znázornění modelování jednotlivých tříd pomocí dvourozměrných Gaussových rozdělení je vyobrazena na obrázku 8.



Obrázek 9. Ilustrace modelování jednotlivých tříd pomocí dvourozměrných Gaussových rozdělní. Převzato z [12].

Funkci si můžeme ilustrovat na příkladě, kde jsou jednotlivé slova popsané celým vektorem, předcházející výsledné rozložení se změní pouze v tom, že je počítáno s dalšími n-rozměrnými Gaussovými rozloženými. Avšak jednotlivé slova jsou reprezentována celou sekvencí 13 resp. 39 rozměrnými vektory. Nabízí se jednoduché řešení, vytvořit pro každý vektor jedno Gaussovské rozložení. Pakliže bychom použili tento přístup, došli bychom na problém s rozdílnou délkou slov, je tedy nutné použít jiný způsob.

Řešením je zavedení modelů, u kterých je možné jednotlivé Gaussovské rozložení opakovat. Tímto řešením ve skutečnosti vytvoříme skryté Markovovy modely (HMM). Pak můžeme jednotlivé slova ze slovníku popsat Markovými modely, ve kterých budou logicky umístěné a pospojované Gaussovská rozložení pravděpodobnosti.



Obrázek 10. Schéma skrytého markovova modelu. Převzato z [12]

Jednotlivé funkce hustoty rozložení vysílacích pravděpodobností pro P-rozměrné Gaussovské rozložení se počítají dle vztahu

$$b_j[o(t)] = \prod_{i=1}^{P} N(o(t); \mu_{ji}, \sigma_{ji}) = \prod_{i=1}^{P} \frac{1}{\sigma_{ji}\sqrt{2\pi}} e^{-\frac{[o(t)-\mu_{ji}]^2}{2\sigma_{ji}^2}} \quad . \qquad (18)$$

Vzorec pro výpočet Gaussova rozložení[12].

Díky znalosti výpočtu vysílacích pravděpodobností Gaussových rozložení můžeme vypočítat celkovou pravděpodobnost, že model *M* generuje sekvenci vektorů *O*. Jednotlivé sekvence vektorů *O* se pokusíme rozložit na HMM stavy, a to takovým způsobem, aby tyto získaly největší možnou věrohodnost („likelihood“).

Princip fungování rozpoznávání řeči založení na využití HMM je následující. Nejprve si rozpoznávač natrénuje modely HMM pro slova která chceme rozpoznávat. Následně na vstup rozpoznávače přijde sekvence parametrů slova *O*. Tuto sekvenci aplikujeme na všechny modely ve slovníku za pomoci Viterbiho algoritmu a ten následně generuje věrohodnosti generování *O* jednotlivými modely. Model s největší věrohodností představuje rozpoznané slovo[16].

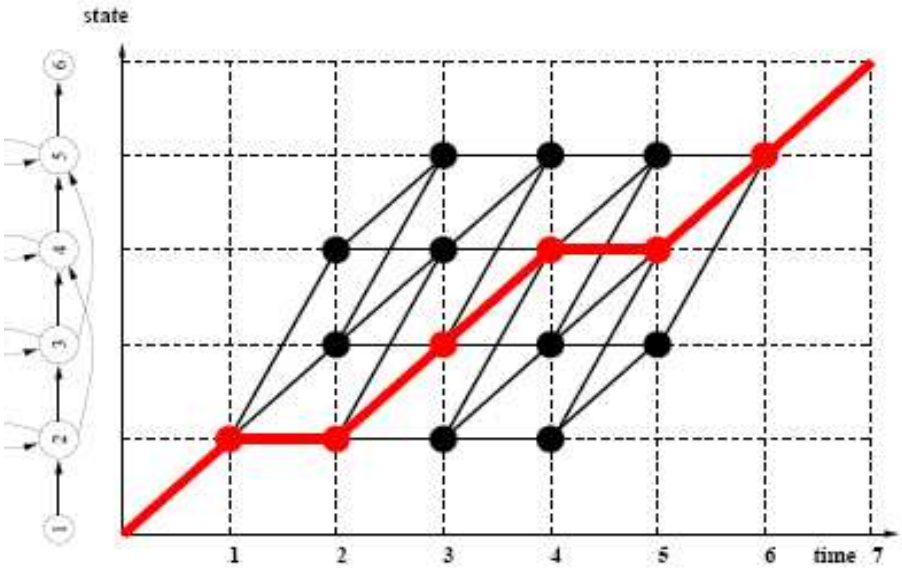
Viterbiho algoritmus pro ohodnocení jednotlivých modelů slov ze slovníku je možné implementovat pomocí velice efektivní metody tzv. „token passingu“. Algoritmus pracuje následujícím způsobem.

HMM tvoří v podstatě automat s vysílajícími stavy a ohodnocenými hranami, jejichž ohodnocení představuje přechodovou pravděpodobnost přechodu z jednoho stavu do jiného. Jelikož HMM pracují s časem, který může plynout jen jedním směrem, není možné přecházet ze stavu s_i do stavu s_{i-1}, jsou umožněny pouze přechody ze stavu s_i do stavu s_{i+1} nebo s_{i+2}, příp. je možné setrvat v původním stavu. Ohodnocené hrany modely tvoří pravděpodobnostní matici s nulami pod diagonálou. Každý stav v sobě obsahuje parametry Gaussova rozložení, jimiž jsou střední hodnota a rozptyl. Tyto parametry určují tvar Gaussovy křivky.

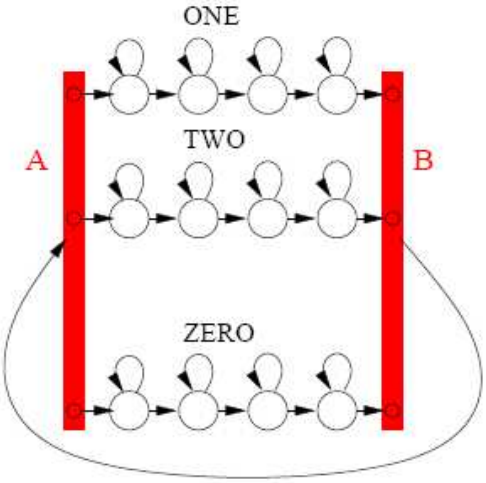
Viterbiho algoritmus hledá rozmístění vstupních vektorů extrahovaných příznaků řeči s nejlepší likelihood, tzn. rozmístění, které bude nejlépe odpovídat původnímu vzoru, v podstatě jde o to najít nejlepší cestu v grafu viz. obrázek 10. Samotný algoritmus token passing funguje tak, že nejprve se každý stav vynuluje. Pak už funguje samotné rozpoznávání. V nultém časovém okamžiku, tj. na vstupu ještě není žádný vstupní vektor příznaků, se umístí do prvního vstupního stavu token s nulovou hodnotou. Po příchodu prvního vektoru, ze vstupního signálu extrahovaných features, dojde k přechodu tokenu ze vstupního stavu do stavu následujícího, přičemž se k jeho hodnotě přičte přechodová pravděpodobnost zvýšená o výstup gaussova rozdělení pravděpodobnosti v příslušném stavu. Po příchodu dalšího vektoru se token rozešle do všech další navázaných stavů přičemž se opět zvýší o přechodovou pravděpodnost a výstup gaussova rozdělení pravděpodobnosti. Pokud se v jednom stavu sejde více tokenů, vybere se ten s největší likelihood a ostatní se zahodí. Tento postup se opakuje tak dlouho, dokud nedojde na vstup poslední vstupní vektor. Poslední vstupní vektor způsobí přechod tokenu do výstupního stavu, přičemž se přičte přechodová pravděpodobnost a takový

token je výslednou hodnotou. Algoritmus se stejným způsobem aplikuje na všechny slova ve slovníku a vítězem se stává ten s největší likelihood.

Výše popsaný algoritmus je možné aplikovat i na rozpoznávání spojené řeči, pouze s tím rozdílem, že výstupní stavy modelů se spojí se vstupními a tokeny tak mohou procházet i několikrát jedním stavem. Schéma tohoto způsobu rozpoznávání je na obrázku 11.



Obrázek 11. Stavová sekvence nejlepšího rozmístění vstupních vektorů[12].



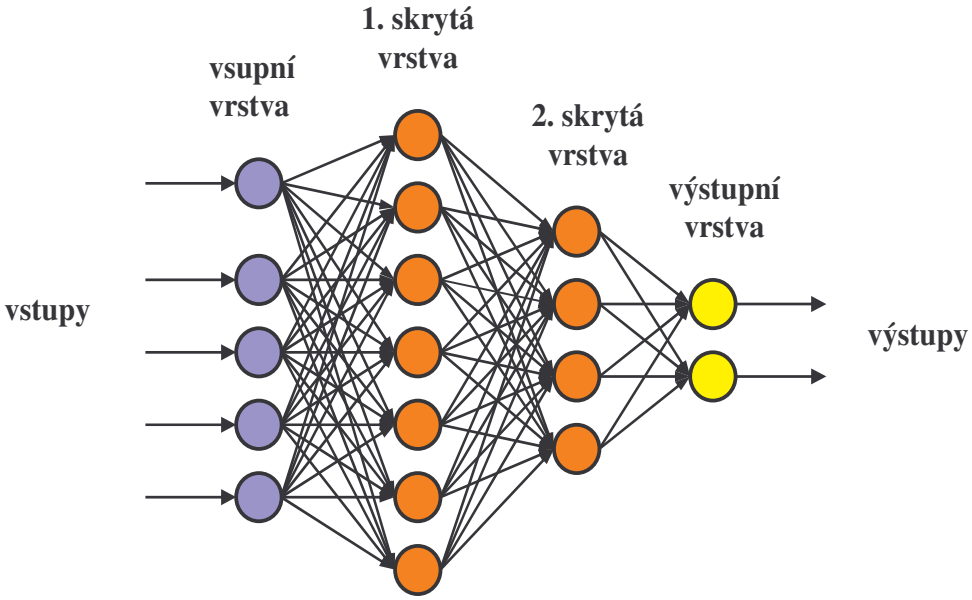
Obrázek 12. Schéma modelu pro rozpoznávání spojených slov[12].

3.3.2.2 Neuronové sítě

Jedním ze široce využívaných metod při rozpoznávání mluvené řeči jsou neuronové sítě. Neuronové sítě jsou jedním z prostředků, který získává své schopnosti učením. Jejich koncepce vychází z biologických neuronových sítí a jejich prvků neuronů. Neurony mají několik vstupů, v závislosti na funkci, a pouze jeden výstup. Každý ze vstupů je ohodnocen svou váhou. Výstup neuronu je generován v závislosti na součtu vahami vynásobenými vstupy a výstupní funkci, která může být realizována několika přenosovými funkcemi.

Aby neurony plnily svou funkci musí být sestaveny do sítě. Neuronové sítě jsou tvořeny neurony poskládanými a propojenými do několika vrstev. První vrstva má funkci vstupní vrstvy, tj. na vstupy neuronů této vrstvy jsou připojeny vstupní signály. Další jedna, případně více vrstev, tvoří tzv. skrytou vrstvu a výstup takovéto neuronové sítě obstarává vrstva poslední, která má za úkol předání výstupní hodnoty vygenerované průchodem vstupních hodnot neuronovou sítí na výstup.

Neuronové sítě jsou dvou typů, síť s dopředným šířením (feedforward neural network), tj. síť kde vstupy jedné vrstvy jsou připojeny na výstupy předchozí vrstvy, a síť se zpětnou vazbou. Sítě s dopředným šířením jsou výhodné zejména proto, že poskytují výstup s konečnou délkou odezvy na jim předložená vstupní data, avšak zásadní nevýhodou je, že tyto sítě se neumějí samostatně učit tzv. „bez učitele“. Naproti tomu sítě se zpětnou vazbou umožňují, aby se systém učil sám „bez učitele“ a tudíž je možné je aplikovat i na řešení úloh, při kterých se dynamický mění podmínky, za kterých má neuronová síť fungovat. V případě použití sítí s dopředným šířením by bylo nutné takovou sít při každé změně podmínek znova přeučit na podmínky aktuální. Nevýhodou sítí se zpětnou vazbou je možné nekonečné zvyšování výstupní amplitudy a následné divergence, při které sít ztrácí schopnost plnit svou funkci.



Obrázek 13. Schéma neuronové sítě s dopředným šířením.

Aby neuronová síť mohla plnit svou funkci je potřeba ji nejprve „naučit“ nutné znalosti. Při učení neuronových sítí se používají dva přístupy. Pro sítě s dopředným šířením se využívá učení s učitelem pro sítě se zpětnou vazbou se využívá učení neuronové sítě bez učitele.

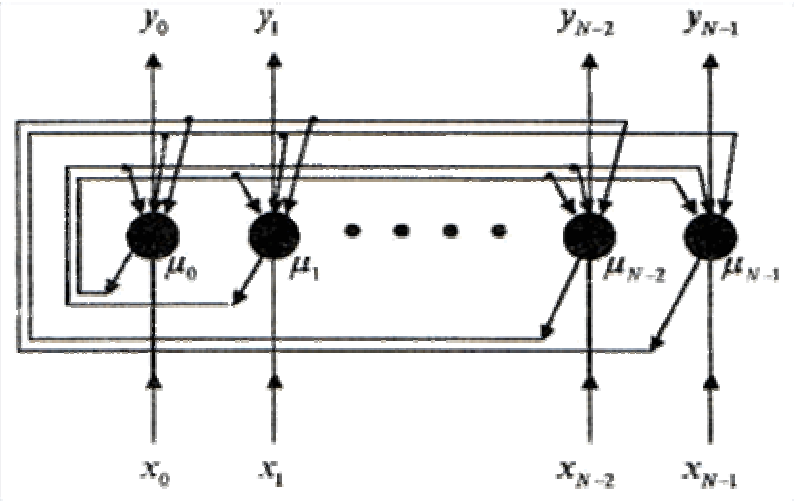
Při učení neuronové sítě s učitelem probíhá trénovaní, tak že trénovaná neuronová sít porovnává svůj výstup, který generuje na základě vstupu a aktuálně nastavených vah na synapsích jednotlivých neuronů, s výstupem, který generuje učitel jež má k dispozici množinu vzorů ve tvaru vstup-výstup. Po každém průchodu vstupních hodnot z množiny trénovacích dat, které předkládá učitel neuronové síti, dojde k porovnání výsledku a pomocí algoritmů určených k učení neuronové

sítě se upraví váhy na jednotlivých vstupech jednotlivých neuronů. Tento proces je iterační, je tedy potřeba pro každou dvojici vstup-výstup provést několik vyhodnocení a úpravu vah, dokud není výstup neuronové sítě v jistém chybovém intervalu vůči požadovanému vzoru nebo dokud nedojde k překročení maximálního počtu iterací.

Pro učení s učitelem se využívá gradientní metody, protože již při malém počtu vstupních hodnot dochází k exponenciálnímu růstu prostoru řešení a je tedy nutné používat efektivnější metody pro hledání řešení. Cílem gradientní metody je minimalizace chyby generované výstupní vrstvou neuronové sítě. Algoritmus, který využívá gradientní metody se nazývá Back propagation.

Princip tohoto algoritmu je víceméně jednoduchý. Algoritmus se pokusí změnit vstupní váhu ne jedné ze synapsí a kontroluje jestli došlo ke zlepšení výstupní hodnoty neuronové sítě, pokud došlo ke zhoršení jde algoritmus opačným směrem. Tento algoritmus má sklony k uvíznutí v lokálních minimech funkcí, pokud k tomu dojde je nutné změnit náhodně několik vah, čímž by mělo dojít k překonání lokálního minima a pokračovat v dalším ohodnocování.

Neuronové sítě se zpětnou vazbou nepoužívají k učení pomoci učitele, ale jsou schopny, se na základě vyhledávání vzorků s určitými vlastnostmi ve vstupních datech a korelací jejich závislosti, učit samy. Příkladem takovéto sítě je síť Hopfiledova, jejíž schéma na obrázku 11. Z tohoto schématu je patrné, jak jsou jednotlivé výstupu neuronů propojený se vstupy.



Obrázek 14. Model Hopfieldovy neuronové sítě. [24]

Neuronové sítě poskytují aplikacím určeným k rozpoznávání řeči široké možnosti použití. Jejich použití je vhodné v aplikacích jejichž vstup je silně ovlivňován rušivými elementy, zejména pak šumem. Neuronové sítě jsou schopné s takovýmito problémy se vypořádat, zejména pak, pokud hovoříme o sítích se zpětnou vazbou, které se mohou přizpůsobovat i dynamicky se měnícím podmínkám, za kterých daná neuronová síť musí pracovat.

Nevýhodou neuronových sítí je jejich náchylnost na generování špatných výsledků vlivem i malých změn vstupních hodnot, způsobených například zaokrouhlováním. Dalším problémem může

být výpočetní náročnost procesu učení, který jako takový je iterativní a na jeden trénovací vstup musí být provedeno několik vyhodnocení kompletní neuronové sítě.

Neuronové sítě jsou díky svému rozsahu, který je nutný abychom dostali přijatelné výsledky, velmi problematicky implementovatelné do hardware jako jsou programovatelná hradlová pole (FPGA). Počet v nich integrovaných prvků je i při současných vysokých hodnotách nedostačující pro implementaci rozsáhlejších neuronových sítí. Podrobněji je problematika implementace neuronových probrána v [25].

4 Rozpoznávání řeči v hardwaru

Aby bylo možné efektivně implementovat algoritmy ve zvolených výpočetních prostředcích je nutné, podrobně znát vlastnosti a architektury těchto výpočetních prostředků.

Následující kapitola pojednává o technologii digitálních signálových procesorů, dále jen DSP, a jejich architekturách, dále je zde popsána technologie programovatelných hradlových polí, dále jen FPGA.

V kapitole budou taktéž diskutovány klady a zápory DSP a FPGA, a případná implementační úskalí, která by mohla nastat při budoucí implementaci algoritmů pro rozpoznávání řeči. Na závěr této kapitoly budou zmíněny dostupné vývojové jazyky a prostředky umožňující vývoj pro danou platformu.

4.1 Možné implementační architektury

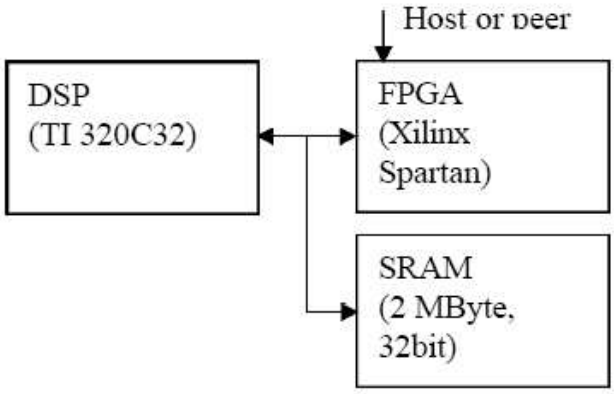
Algoritmy rozpoznávání řeči patří k algoritmům s vysokými výpočetními i paměťovými nároky, a proto je vhodné hledat způsoby, které by umožnily jejich funkční implementaci i bez toho, aby bylo nutné využívat výkonné a tudíž i energetický náročné procesory, které by zvláště v případě aplikací, jakými jsou mobilní telefony nebo domácí spotřebiče, znamenaly výrazné snížení výdrže baterií a v neposlední řadě je třeba brát v potaz také finanční hledisko, které je mnohdy určujícím kritériem pro volbu daného zařízení, zejména pak s ohledem na budoucí energetickou krizi, která Nás pravděpodobně nemine.

Vhodným řešením je využít digitální signálové procesory, které jsou určeny pro zpracování audio i jiných signálů. Avšak DSP procesory vesměs neposkytují tolik výkonu, alespoň pokud požadujeme nízkou spotřebu, jaký by byl potřeba při rozpoznávání řeči, proto je vhodné využít hardwarové akcelerace založené na použití programovatelných hradlových polí (FPGA). FPGA poskytují vývojářům vysoký výpočetní výkon, který je dán tím, že jednotlivé algoritmy jsou implementovány přímo na čipu a tudíž odpadá režie, která je nutná v případě univerzálních procesorů. Dalším faktorem přispívajícím k vysokému výkonu je značná míra paralelizace, která vyplývá ze samotné podstaty funkce HW. Vysoký výkon FPGA na druhou stranu neomezuje případný budoucí update implementovaných algoritmů, neboť FPGA je možné přeprogramovat, sofistikovanější modely i za chodu, aniž by byla ovlivněna jejich funkce.

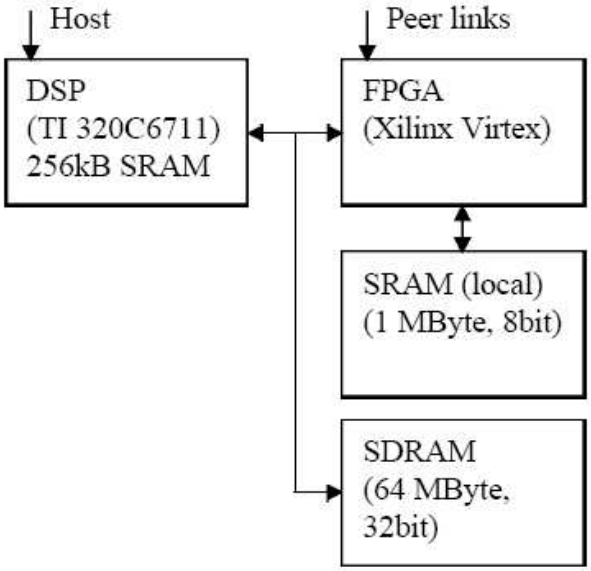
4.1.1 DSP/FPGA platforma

Výhody DSP a FPGA na jedné platformě využívá i návrh, který byl vyvinut na FIT VUT v Brně, a který je primárně využit pro akceleraci algoritmů z oblasti počítačové grafiky.

Platforma byla vyvinuta ve více verzích s rozdílnými HW prostředky. První z nich byla založena na signálovém procesoru firmy Texas Instruments TMS320C32 a hradlovém poli od firmy Xillinx Spartan, ta vznikla v roce 1997. Schéma je vyobrazeno na obrázku 11. Tato platforma byla vybavena 2MB RAM a poskytovala dostatečný výkon pro akceleraci 2D grafických algoritmů. Aby bylo možné využívat platformu i pro aplikace z 3D grafiky, bylo nutné vytvořit inovovanou verzi. Tato novější platforma obsahuje DSP TMS320C6711 a FPGA Xillinx Virtex, kombinace těchto dvou obvodu je doplněna o 64MB dynamické paměti, díky čemuž tato platforma poskytuje dostatečný výkon pro 3D aplikace. Její schéma je na obrázku 12.



Obrázek 15. Schéma platformy s TMS320C32 a Xillinx Spartan. Převzato z [23].



Obrázek 16. Schéma platformy s TMS320C6711 a Xillinx Spartan. Převzato z [23].

4.1.1.1 Specifikace DSP

Popisovaná platforma využívá signálového procesoru firmy Texas Instruments, podle verze to může být procesor TMS320C32 nebo TMS320C6711. Oba uvedené signálové procesory pracují s floating-point aritmetikou a jsou vhodné pro zpracování grafických algoritmů.

Specifikace TMS320C32

Specifikace signálového procesoru TMS320C32 dle [18].

- vysoce výkonný floating-point DSP procesor
- 330MOPS
- 60MFLOPS
- 30MIPS
- 32 bitový procesor
- 32 bitové instrukční slovo s 24 bitovou adresou
- EMIF s podporou 8/16/32/ bitové externí RAM

Specifikace TMS320C6711

Specifikace signálového procesoru TMS320C6711 dle [19].

- vysoce výkonný signálový procesor
- architektura VLIW s možnosti zpracování až osm 32 bitových instrukcí v jednom cyklu
- frekvence 100, 150, 167, 200MHz při výkonu 600, 900, 1000, 1200MFLOPS
- 4x ALU (Fixed- a Floating-Point), 2x ALU (Fixed-Point), 2x násobička (Floating- a Fixed-Point)
- L/S architektura s třicetidvěma 32 bitovými registry
- L1 cache 4kB pro data, 4kB pro program
- L2 cache 512kB s mapováním do RAM

4.1.1.2 FPGA

DSP procesoru sekunduje FPGA čip z rodiny Xillinx Spartan, to v případě starší verze, příp. Xillinx Virtex, to v případě novější verze platformy. Podrobnější popis platformy FPGA lze najít v [20] a popis přesných specifikací viz. [21], [22].

Specifikace Xillinx Spartan

- nízko nákladové, vysoce výkonné hradlové pole
- až 74 880 logických buněk
- rychlost přenosu dat až 622Mb/s pře I/O
- podpora DDR

Specifikace Xillinx Virtex

- počet hradel od 50000 do 1mil.
- frekvence hodin až 200MHz
- hierarchická paměť
- možnost parciální rekonfigurace

4.1.2 DSP vývojový kit Spectrum Digital

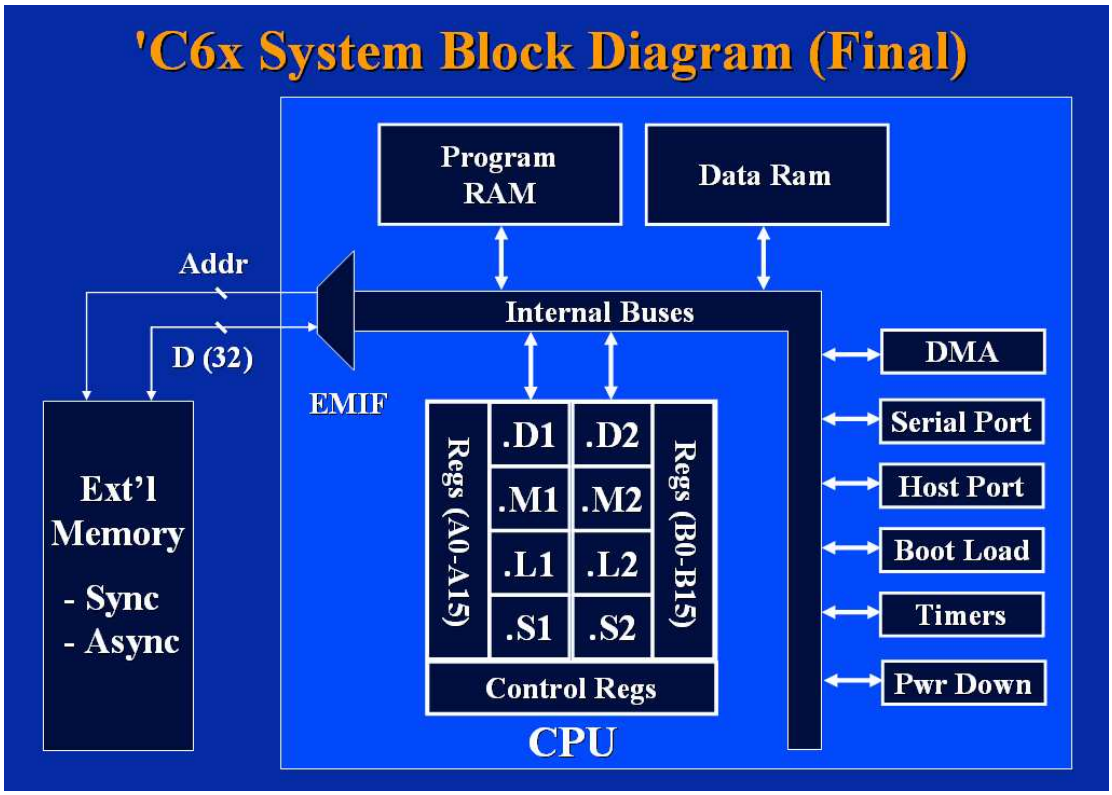
Vývojový kit firmy Spektrum Digital je určený pro vývoj vestavěných aplikací využívajících signálového procesoru od firmy Texas Instruments TMS320C6416, který je primárně určen pro zpracování zvukových signálů, pro které je optimalizován. Signálový procesor je optimalizován pro své určení na úrovní instrukční sady a architektury. Zvukový signál je vzorkován a kvantizován, a tedy jeho hodnoty jsou uloženy v celých hodnotách, tudíž procesor využívá pouze celočíselnou aritmetiku, která je značně rychlejší než výpočty v plovoucí řádové čárce. Nicméně díky podpoře kompilátoru je možné využívat výpočtů v plovoucí řádové čárce, které jsou na tomto DSP emulovány. Přirozeně nevýhodou tohoto způsobu je výrazně nižší počet operací proveditelných za jednotku času.

Architektura procesoru je typu VLIW(Very Long Instruction Word), tudíž používá dlouhá instrukční slova, ve který jsou zakódovány instrukce pro 8 nezávislých funkčních jednotek, 2 násobičky a 6 aritmeticko-logických jednotek.

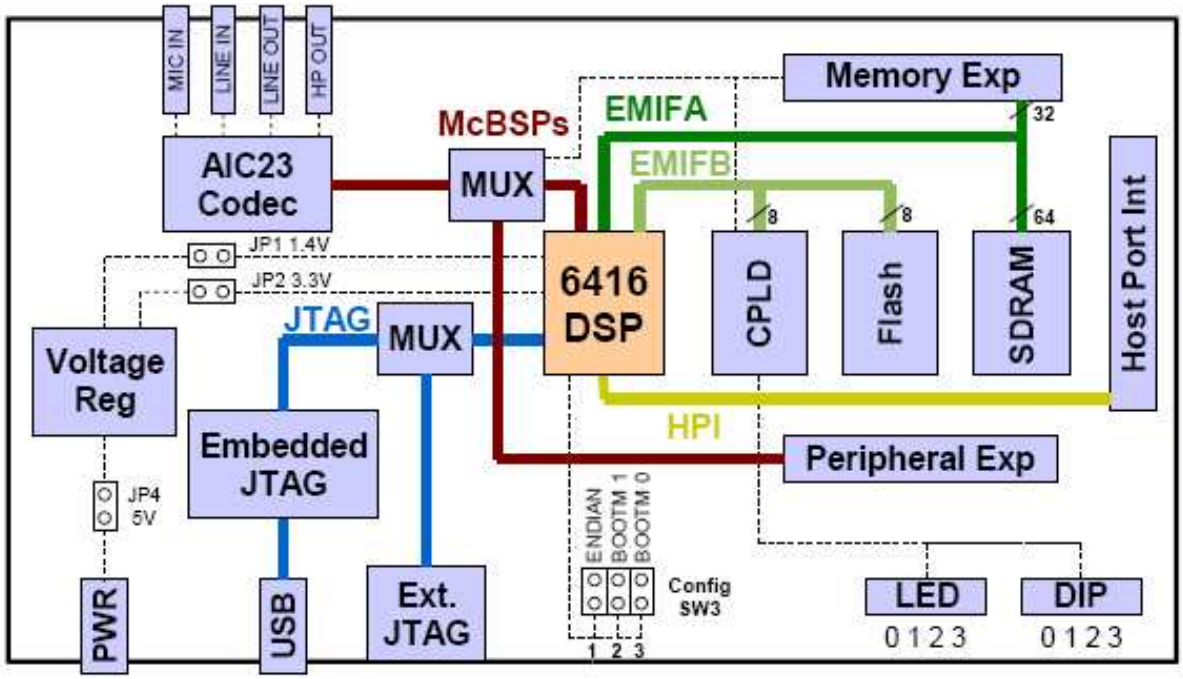
4.1.2.1 Architektura vývojového kitu a DSP

Schéma architektury a vnitřní struktury procesoru je zobrazeno na obrázku 17. Ze schématu je patrné jak jsou uspořádány registry procesoru, a jak jsou napojeny na funkční jednotky. Procesor má dvě sady registrů, což je oproti konvenčním univerzálním procesorům zásadní rozdíl. Toto řešení umožňuje zvýšení výkonnosti podpořené velkým množství různých adresovacích režimů, které jsou speciálně určené pro zpracování specifických úloh, jako je výpočet rychlé Fourierovy transformace. Dalším specifikem je použití Harvardské koncepce, která používá oddělené paměťové prostory pro program a data. Komunikace s okolními prostředky a rozhraními je umožněna díky vnitřní sběrnici, na kterou jsou všechna tato zařízení připojena. Komunikace s externí dynamickou pamětí je umožněna díky EMIF rozhraní.

Schéma zapojení periférií na desce vývojového kitu je vyobrazeno na obrázku 18. Deska má integrovány periferie pro komunikaci s dynamickou pamětí, rozhraní AIC23 určené k digitalizaci zvuku z externích zdrojů, JTAG rozhraní pro komunikaci přes rozhraní USB, sadu LED diod a sadu DIP přepínačů. Přes rozšiřující rozhraní UTOPIA je možné podle potřeby připojovat další zařízení v závislosti na budoucích aplikacích DSP procesoru.



Obrázek 17. Vnitřní architektura procesoru.



Obrázek 18. Schéma zapojení periférií na vývojovém kitu.

4.1.2.2 Softwarová podpora

Nedílnou součásti vývojového kitu je také softwarová výbava, která poskytuje velice silnou podporu vývoje vestavěných aplikací.

Základními stavebními kameny jsou DSP/BIOS a vývojové prostředí Code Composer Studio.

DSP/BIOS je v podstatě takovým operačním systémem DSP, který má za úkol správu a konfiguraci všech externích zařízení, správu dynamické paměti a její přidělování běžícím procesům. Dále má na starosti přidělování procesorového času procesům, které jsou iniciovány na základě přerušovacích rutin, které mají velice silnou podporu, protože aplikace pro zpracování signálů musí dodržovat jisté real-time podmínky. Přidělování procesů je plánováno na základě příslušných real-time plánovacích algoritmů, které přidělují procesor na základě priorit procesů a časových mezí, které musí být splněny.

Samotný vývoj aplikací probíhá pomocí vývojového prostředí Code Composer Studio, které poskytuje v jednom aplikačním prostředí podporu pro konfiguraci všech rozhraní dostupných na vývojové desce, dále obsahuje kompilátor s možností vysoké míry optimalizace vzhledem k požadavkům budoucí aplikace, bohatou dokumentaci s velkým množstvím příkladů a velkým množstvím podpůrných funkcí určených pro ladění vyvíjených aplikací. Vzhled aplikace je vyobrazen v příloze A.

5 Návrh

Tato kapitola popisuje návrh algoritmů pro rozpoznávání řeči, možné přístupy k návrhu a dekompozici problému na vybrané platformě.

5.1 Specifikace požadavků

Cílem této práce je vytvořit rozpoznávač jednotlivých slov, který bude implementován na vestavěném systému. Tímto vestavěným systémem bude technologie popsána v kapitole 4.. Implementovaný rozpoznávač má využívat výhod kombinace DSP a FPGA, přičemž FPGA bude zajišťovat akceleraci vybraných algoritmů, jež se při rozpoznávání slov uplatňují. Při návrhu je potřeba brát v úvahu vlastnosti vybraného systému vzhledem k použitým algoritmům, zejména pak nižší výpočetní výkon, značně omezené paměťové možnosti, omezenou složitost algoritmů z pohledu použitelných zdrojů na FPGA čipu a v neposlední řadě omezené množství času jež je k dispozici k řešení zadaného problému.

Rozpoznávač by měl umět na základě vstupu klasifikovat slova obsažená ve slovníků, tato slova budou popsána Gaussovskými pravděpodobnostmi, na základě kterých budou slova rozdělena do tříd skrytých Markovových modelů.

Rozpoznávány budou slova z omezeného slovníku, který bude obsahovat prozatím číslovky v českém jazyce, tato volba není finální, neboť zde hraje roli množství paměti potřebné pro dostatečně kvalitní popis jednotlivých tříd slov a tudíž je možné, že množství slov se může změnit, a to oběma směry.

Vstup bude realizován jako zvukový soubor nebo vstup z externího zdroje zvuku.

Výstupem bude přiřazení vzoru ke vstupnímu signálu.

5.2 Návrh postupu zpracování

Zpracování vstupního signálu, ať už ve formě soboru nebo ve formě audio, bude probíhat na základě posloupnosti kroků, která je probrána v následující podkapitole.

5.2.1 Parametrizace (Extrakce příznaků)

Základem rozpoznávání řeči je extrakce příznaku ze vstupního signálu. Pro rozpoznávání v embeded systému jsme zvolili jako parametry Melfrekvenční cepstrální koeficienty, které jsou popsány v kapitole 3.3.

Vstupem systému bude, jak už bylo uvedeno, soubor nebo vstupní signál. U tohoto signálu předpokládáme vzorkovací frekvenci $F_s=8kHz$ nebo $F_s=16kHz$, která se pro tyto účely používá, protože frekvence běžné řeči se pohybuje pod $F_{max}=4kHz$, což po započítání požadavku na dvojnásobnou vzorkovací frekvenci vyplývající z Nysquitova vzorkovacího teorému, dává výše uvedenou vzorkovací frekvenci.

V následujícím kroku potřebujeme získat frekvenční spektrum daného rámce, proto na daný rámec aplikujeme rychlou Fourierovu transformaci (Fast Fourier Transform – FFT), která vypočte spektrum definované komplexními čísly pro daný rámec. Z těchto komplexních čísel vypočteme moduly a umocníme na druhou, z čehož získáme power spektrum. Následně aplikujeme banky 23 Mel-filtrů popsaných v kapitole 3.3.1.3. Z hodnot získaných aplikací filtrů dostaneme MFCC koeficienty tak, že aplikujeme diskrétní kosínovu transformaci, na jejímž výstupu bude 13 MFCC parametrů.

5.2.2 Modely rozpoznávaných slov

Rozpoznávání slov založené na porovnání se vzory vyžaduje mít tyto vzory nějakým způsobem uloženy ve slovníku. V případě rozpoznávače implementovaného pro účely této práce je rozpoznávání založeno na statistickém modelování za pomocí skrytých Markovových modelů. Tedy slova ze slovníku jsou reprezentována HMM.

Při tvorbě modelů budeme uvažovat počet stavu HMM za pevně daný. S ohledem na rozpoznávání jednotlivých číslovek zvolíme počet stavů HMM na 16, k těmto stavům je nutné uvažovat i stavy počáteční a koncové, tudíž je použito celkem 18 stavů.

Jednotlivé stavy jsou reprezentovány Gaussovými rozloženími pravděpodobnosti, přičemž použijeme reprezentaci pouze jedním Gaussovským rozložením.

Trénování modelů rozpoznávaných slov bude prováděné na počítači, z důvodu značných výkonnostních a paměťových omezení.

5.2.3 Algoritmus rozpoznávání

Navrhovaný rozpoznávač jednotlivých slov v FPGA/DSP využívá statistického modelovaní pomocí HMM. Základním stavebním prvkem jsou slova, jež jsou jako model uložena ve slovníku. Rozpoznávač má za úkol určit model s největší věrohodností, který definuje rozpoznané slovo.

Pro proces rozpoznávání bude vhodné využít, široce používaného, Viterbiho algoritmu, jež na vstupu bude mít MFCC koeficienty, které se získají výše uvedeným postupem a skryté Markovovy modely. MFCC koeficienty jednotlivých rámců se postupně porovnávají se všemi modely slov ve slovníku a výsledky porovnání se budou průběžně ukládat. Po zpracování části signálu, který obsahuje slovo, dojde k vyhodnocení a určení slova s největší věrohodností.

Při samotném rozpoznávání je jednou z výpočetně nejobtížnějších části výpočet gaussova rozloženi pro každý vektor a stav skrytého Markovova modelu. Z rovnice 17 je zřejmé, že při výpočtu je nutné použít výpočtu exponentu, což je výpočetně náročná operace a její provádění v reálném čase, by pro značný počet stavů a modelů bylo prakticky nerealizovatelné, proto je třeba hledat způsoby jakými by bylo možné výpočet vysílacích pravděpodobnosti stavu $b_j[o_t]$ zrychlit. Jednoduchým způsobem je zlogaritmování celé rovnice, čímž převedeme násobení na pouhé sčítání a zbavíme se výpočetně náročného exponentu. Odvození patřičných rovnic je uvedeno v rovnici 18.

$$\log b_j[o(t)] = \log \prod_{i=1}^{P} \mathrm{N}(o(t); \mu_{ji}; \sigma_{ji}) = \log \prod_{i=1}^{P} \frac{1}{\sigma_{ji}\sqrt{2\pi}} \cdot e^{-\frac{[o(t)-\mu_{ji}]^2}{2\sigma_{ji}^2}} =$$

$$= \log \left[\prod_{i=1}^{P} \frac{1}{\sigma_{ji}\sqrt{2\pi}} \times \prod_{i=1}^{P} e^{-\frac{[o(t)-\mu_{ji}]^2}{2\sigma_{ji}^2}} \right] = \log \prod_{i=1}^{P} \frac{1}{\sigma_{ji}\sqrt{2\pi}} + \log \prod_{i=1}^{P} e^{-\frac{[o(t)-\mu_{ji}]^2}{2\sigma_{ji}^2}}$$

Členy v součtu v rovnici je dále možné zjednodušit na následující dvě rovnice.

$$\log \prod_{i=1}^{P} \frac{1}{\sigma_{ji}\sqrt{2\pi}} = \log \frac{1}{\prod_{i=1}^{P} \sigma_{ji} \cdot (2\pi)^{\frac{P}{2}}} = -\log \left(\prod_{i=1}^{P} \sigma_{ji} \cdot (2\pi)^{\frac{P}{2}} \right) =$$

$$= -\frac{1}{2} \log \left(\prod_{i=1}^{P} \sigma_{ji} \cdot (2\pi)^{P} \right)$$

$$\log \prod_{i=1}^{P} e^{-\frac{[o(t)-\mu_{ji}]^2}{2\sigma_{ji}^2}} = \sum_{i=1}^{P} \log \left(e^{-\frac{[o(t)-\mu_{ji}]^2}{2\sigma_{ji}^2}} \right) = \sum_{i=1}^{P} -\frac{[o(t)-\mu_{ji}]^2}{2\sigma_{ji}^2} =$$

$$= -\frac{1}{2} \sum_{i=1}^{P} \left(\frac{o(t)-\mu_{ji}}{\sigma_{ji}} \right)^2$$

Výslednou rovnici pak můžeme zapsat ve tvaru:

$$\log b_j[o(t)] = -\frac{1}{2} \log \left(\prod_{i=1}^{P} \sigma_{ji} \cdot (2\pi)^{P} \right) - \frac{1}{2} \sum_{i=1}^{P} \left(\frac{o(t)-\mu_{ji}}{\sigma_{ji}} \right)^2 =$$

$$= -\frac{1}{2} \left(\log \left(\prod_{i=1}^{P} \sigma_{ji} \cdot (2\pi)^{P} \right) + \sum_{i=1}^{P} \left(\frac{o(t)-\mu_{ji}}{\sigma_{ji}} \right)^2 \right) \qquad (18)$$

Z výsledné rovnice je patrné, že proměnnou hodnotou je pouze vstupní vektor příznaků *o(t)*. Můžeme si tedy dopředu předpočítat hodnotu **gConst**, kterou získáme z první části předešlé rovnice.

$$gConst = \log\left(\prod_{i=1}^{P}\sigma_{ji}\cdot(2\pi)^{P}\right) \qquad (19)$$

5.3 Návrh dekompozice problému mezi FPGA/DSP

Návrh rozpoznávače musí využívat výhod kombinace FPGA a DSP integrovaných na jedné desce, proto je nezbytné co možná nejlépe dekomponovat rozpoznávání na jednotlivé úlohy a ty přiřadit příslušným jednotkám.

Pro implementaci předpokládám, že DSP bude zajišťovat komunikaci s hostitelským počítačem. Vzhledem k tomu, že signálový procesor implementovaný na použité platformě se využívá v současných implementacích pouze jako prostředník pro komunikaci mezi FPGA a hostitelským počítačem, bude samotné vzorkování signálu a extrakce příznaků ponechána na hostitelském počítači, který bude zasílat pouze vektor s již vypočtenými příznaky. Přirozeným způsobem by bylo, aby extrakce příznaků byla implementována přímo v DSP, ale v tomto případě by to značilo kompletně přepracovat software zabudovaný v DSP, který má na starosti řízení a programování FPGA. Proto bude lepší využít již existujících kódů zajišťujících přístup k FPGA. S ohledem na výkonnost systému je použití DSP pouze pro programování a komunikaci s DSP vhodným řešením, protože urychlení MFCC není příliš přínosné, neboť nejnáročnější částí celého rozpoznání je ohodnocení modelů.

Samotné FPGA bude zajišťovat vyhodnocení HMM pomocí Viterbiho algoritmu. Platforma jako taková se sestává ze základní desky, na kterou je možné umístit až čtyři moduly integrující FPGA a DSP, a tedy je možné do každého modulu nahrát jeden model a pak všechny tuto modely vyhodnotit paralelně. Jelikož FPGA pracuje pouze s čísly v pevné řádové čárce musí se data před vyhodnocováním v FPGA převést na celočíselnou reprezentaci, což bude mít za následek zhoršení schopnosti evaluace HMM modelů.

Rozdělení úloh mezi FPGA a DSP je pouze hrubým návrhem, neboť dosud nejsou známé požadavky na zdroje pro jednotlivé algoritmy, zejména počet hradel na FPGA, který je značně omezujícím faktorem. Pakliže by se předpokládané algoritmy nepodařilo vhodně umístit do FPGA, je možné tyto algoritmy implementovat v DSP. V příloze B je schéma rozvržení FPGA a DSP na desce.

Protože cílem této diplomové práce je zhodnotit možnosti implementace algoritmů v FPGA/DSP výpočetních prostředcích je možné, že se během implementace vyskytnou překážky,

které by znemožnily úspěšné dokončení implementace na vybrané platformě. Pro tento případ bude použito náhradní řešení ve formě implementace rozpoznávače na vývojové platformě, která využívá pouze DSP procesor. Pro tyto volbu implementace je možné využít stejného návrhu jako pro rozpoznávač na FPGA/DSP.

6 Implementace

Tato kapitola se zabývá samotnou implementací rozpoznávače v HW, popisuje nutná nastavení hardwarových prostředků, popis samotné implementace, budou zde také diskutovány problémy, které vznikly během implementace a popis datových struktur a jejich inicializace. Součásti kapitoly budou průběžná hodnocení výkonnosti a zátěže DSP při výpočtech.

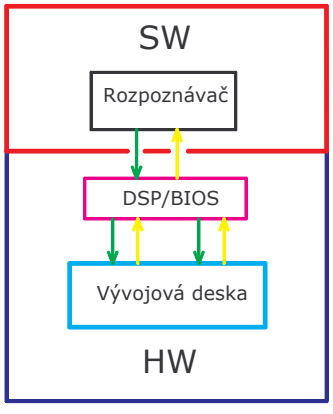
6.1 Výběr implementační platformy

Cílem této diplomové práce bylo implementovat rozpoznávač na platformě kombinující FPGA a DSP, běžně užívané k akceleraci grafických výpočtů. Po konzultacích s Ing. Žadníkem a zhodnocení časových možností jsem dospěl k závěru, že vytvoření funkční aplikace na této platformě je v daném časovém horizontu nereálné, zejména také proto, že neexistuje jakákoliv použitelná dokumentace, ze které by bylo možné vycházet. Jedinou dostupnou dokumentací je schéma zapojení jednotlivých obvodů a jejich propojení signály. Dle mého odhadu by na vypracování rozumné aplikace bylo potřeba odhadem 1-2 roky.

Jako náhradní řešení jsem vybral implementaci jednoduchého rozpoznávače na vývojovém kitu firmy Spectrum Digital. Tato implementace bude popsána v následujících odstavcích.

6.2 Hierarchie rozpoznávače

Implementovaný rozpoznávač využívá vzájemné součinnosti několika úrovní. Úrovně je možné rozdělit do dvou základních tříd a to úroveň hardwarovou a softwarovou. Hardwarová úroveň má za úkol obstarávat konfiguraci zařízení a obsluhu jejich přerušení s následným generování softwarových přerušení pro softwarovou vrstvu. Na této úrovni uvažujeme periferní zařízení, tj. AIC23 kodek, LED diody a DIP přepínače, a DSP/BIOS. Softwarová úroveň provádí samotný program a generuje požadavky na hardwarová zařízení. Hierarchii je možné vidět na následujícím obrázku.



Obrázek 19. Hierarchie rozpoznávače.

6.3 Konfigurace hardwarových zařízení

Implementace jakékoliv aplikace v hardwarových prostředcích má oproti běžnému vývoji aplikací na univerzálních výpočetních systémech několik specifik. Jedním a patrně nejpodstatnějším je přímý přístup k hardwarovým prostředkům, ať už se jedná o periferní zařízení nebo přístup k paměti. Tyto prostředky je nutné před započetím samotné práce nejprve nakonfigurovat. K tomuto slouží konfigurační soubor *dsk_app.cdb* jež obsahuje nastavení pro všechna zařízení jež může vývojová deska využívat. Samotný soubor naštěstí není nutné kompletně konfigurovat ručně, neboť k tomuto slouží grafické rozhraní, které je součástí Code Composer Studia. Ruční konfigurace je však samozřejmě možná. V implementaci rozpoznávače budou využity tyto hardwarové prostředky, které je nutné nakonfigurovat: správa dynamické paměti, AIC23 kodek použitý na vzorkování vstupního signálu přicházejícího z mikrofonu, logovací objekt určený k výpisu ladících a informačních dat, správa hardwarových a softwarových přerušení.

6.3.1 Dynamická paměť

Funkce rozpoznávače využívá relativně velké množství paměti pro uložení modelů slov, a pro uložení a zpracování vstupního signálu. Tato paměť musí být přirozeně někde alokována. Alokace paměti na této platformě je poněkud odlišná od alokace paměti na univerzálním počítači. Pro samotnou alokaci je nejprve nutné nastavit příslušný objekt, který se o ni bude starat. Nastavení se provádí pomocí konfiguračního nástroje obsaženého v CCS. Položku `SYSTEM-MEM-SDRAM` nastavíme následujícím způsobem. Nejprve povolíme vytváření heapu v paměti SDRAM a určíme maximální velikost bloku, který můžeme alokovat. Velikost nastavíme na hodnotu 0x1000000, čímž získáme k dispozici celou dostupnou SDRAM o velikosti 16MB. Aby bylo možné k tomuto bloku paměti přistupovat označíme jej _SEG0. Toto je zásadní rozdíl oproti běžným zvyklostem u univerzálních počítačů.

Je možné také alokovat místo v paměti ISRAM o velikosti 1MB, která je integrována přímo na čipu, avšak tato paměť je primárně určena pro ukládání globálních proměnných a funguje také jako cache paměť.

Pro alokaci paměti se využívá funkce *MEM_alloc*, která je speciálně optimalizována pro potřeby DSP. Její hlavní odlišností od běžně používaného *malloc* je její atomicita, čili funkce nemůže být přerušena jiným procesem, a dále její parametry, kde musí být uveden identifikátor bloku paměti, ve které se má paměť alokovat.

Pro potřeby implementace je v souboru *config.h* vytvořeno makro *mem_alloc*, které na základě počtu bytů provede alokaci příslušné paměti. Po úspěšné alokaci je pomocí logovacího objektu vypsán počet aktuálně alokovaných bytů. Celkový počet bytů alokovaných během funkce aplikace je uložen v globální proměnné *allocatedMem*.

6.3.2 LOG objekt

Aby bylo možné aplikaci nějakým rozumným způsobem ladit, potřebujeme mít možnost získávání ladících výpisů. Běžně používané *printf* je výpočetně hodně náročné, protože neúměrně zatěžuje DSP a USB komunikaci, která sama o sobě má velkou režii. Pro účely výpisu se používá logovací objekt *logTrace*, který se vytvoří v nastavení systému `Instrumentation-LOG`. Pro výpis dat se pak volá funkce *LOG_printf*.

6.3.3 AIC23, McBSP a EMIF

V implementaci rozpoznávače v hardwaru budeme využívat zařízení pro vzorkování vstupního signálu AIC23, sériovou linku McBSP, která bude zajišťovat rychlou komunikaci mezi zařízeními a DSP procesorem, a také rozhraní EMIF, jež je řadičem pro přístup do paměti integrované na desce.

Všechna zařízení musí být nakonfigurována a inicializována před prvním použitím.

Kodek použitý pro vzorkování signálu se nastavuje vytvořením globální proměnné *config*, která je deklarována v souboru s globálními proměnnými *globals.h*. Touto proměnnou nastavíme důležitý parametr, jímž je vzorkovací frekvence, na které bude kodek pracovat, v tomto případě 8kHz. Dále nastavíme vstupní konektor na mikrofonní vstup, přip. linkový vstup. Je možné dále nastavit specifické vlastnosti jako hlasitost jednotlivých kanálů, úsporný režim atd.. Konfiguraci provedeme zavoláním funkce *AIC23_setParams*, které předáme jako parametr proměnnou *config*.

Pro správnou funkci je nutné nastavit a vytvořit komunikační linku, po které bude kodek posílat navzorkovaná data ke zpracování. Nastavení příslušných komunikačních linek se provádí voláním funkce *initMcsb*, která inicializuje linky podle globální proměnné *mcbspCfg2*.

Spolu s konfigurací sériových komunikačních linek se nastavuje také přímý přístup do paměti, který zajistí, že procesor nebude při příjmu dat vytížen, protože data bude možné přímo zapisovat do paměti a až po naplnění příslušných bufferů se vygeneruje přerušení, které bude iniciovat vyvolání příslušné obslužné rutiny. Aby bylo možné data plynule obstarávat data z audio vstupů, používá se pro tyto účely koncept tzv. *PING PONG* bufferů, kdy se do jednoho bufferu data zapisují a druhý buffer je zpracováván. Pro tyto účely jsou vytvořena dvě datová pole *gBufferRcvPing* a *gBufferRcvPong*. Při naplnění jednoho z bufferů se generuje hardwarové přerušení, které iniciuje vyvolání obslužné rutiny *edmaHwi*, které vybere příslušný buffer a jeho obsah předá funkci *processBufferSwi*, která má za úkol provést jeho zpracování. Nastavení DMA řadiče je uloženo v proměnné *gEdmaConfigRcv* a pomocí funkce *initEdma* se nastavení parametrů zapíše do příslušných registrů. Přerušovací rutiny se následně inicializují pomocí funkce *initIrq*.

6.4 Softwarová implementace

Máme-li provedenou potřebnou konfiguraci hardwarových prostředků může přejít k samotné implementaci vlastního rozpoznávače.

6.4.1 Předzpracování signálu

Jak již bylo uvedeno v kapitole 3.3.1.1 musí být signál před samotným rozpoznáním předzpracován, aby se odstranily nežádoucí vlivy, které by mohly negativně ovlivnit proces rozpoznání. Základní předzpracování, tj. ustředění a filtrace preemfázovacím filtrem, má na starosti funkce *oneChanExtrCenterPreem*. Funkce převezme jako parametr pole, které obsahuje vstupní data pořízené audio kodekem, a extrahuje z nich data pro jeden audio kanál, protože příchozí data jsou uložena prokládaně, střídavě pro levý a pravý kanál. V dalším kroku se provede výpočet střední hodnoty signálu a celý signál se ustřední, po této operaci se provede filtrace preemfázovacím filtrem. Při zpracování se uvažuje s překrytím rámců. Překryté vzorky jsou uloženy v poli *continousBuffer*.

6.4.2 Zpracování rámce

Máme-li vstupní signál předzpracován, může se dostat ke slovu funkce *processFrames*, která zajistí rozdělení celého předzpracovaného vstupního bufferu na jednotlivé rámce, jejichž délka je definována pomocí makra *FRAME_OVERLAP* v souboru *config.h*. Při rozdělování na rámce se uvažuje i překryv rámců. Každý vytvořený rámec se následně předá ke zpracování funkci *computeMFCC*.

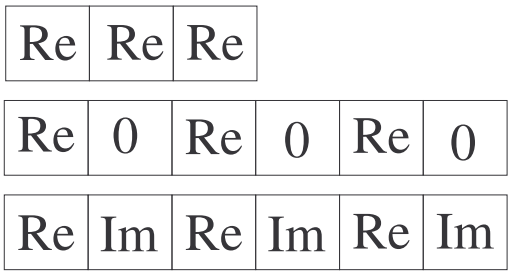
6.4.2.1 computeMFCC

Z rámce, který nám poskytla funkce *processFrames,* potřebujeme vyextrahovat příslušné MFCC příznaky, k tomu slouží funkce *computeMFCC*, která bude teď popsána.

První operací, kterou aplikujeme na vstupní rámec je vynásobení hammingovým oknem. Násobení hammingovým oknem je prosté násobení dvou polí, přičemž koeficienty hammingova okna jsou předpočítany v poli *hammingVec*.

Dalším krokem při získávání MFCC koeficientů je výpočet spektra rámce pomocí FFT. Pro její výpočet použijeme funkci *DSP_fft32x32*, která dostupná v knihovně *dsp64x.lib*. Funkce z této knihovny je implementována s důrazem na maximální efektivitu. Základem pro výpočet FFT je předpočítání tzv. twiddle vektoru, což jsou hodnoty komplexní exponencíály vypočítané pro daný počet bodů FFT. Počet bodů FFT je určen v souboru *config.h* makrem *FFT_VEC_LENGTH*. Twiddle vektor je uložen v globálním poli *twiddleVector* a je inicializován při startu aplikace voláním funkce *genTwiddle*. Funkce pro výpočet FFT počítá s reálnou i komplexní složkou, proto je třeba vstupní vektor upravit tak, aby obsahoval obě složky. Čísla jsou ukládána střídavě ve tvaru reálná složka-

komplexní složka. Patřičnou transformaci zajistí funkce *real2Complex*, jež vezme vstupní rámec a roztáhne jej na dvojnásobnou délku, přičemž každá imaginární složka má hodnotu nula. Schéma převodu je znázorněno na obrázku 20..



Obrázek 20. Schéma uložení komplexních čísel.

Výstupem FFT je spektrum daného rámce, avšak pro výpočet MFCC potřebujeme znát power spektrum, což je součet druhých mocnin jednotlivých složek komplexního čísla. Převod spektra na power spektrum zajišťuje funkce *complex2realPower*, jak je možné vyčíst z jejího názvu, tak se také funkce stará o zpětný převod z komplexní reprezentace do reálné reprezentace.

MFCC koeficienty využívají transformace na mel-frekvenční stupnici. Jednoduchým řešením jak toto zajistit je mít vygenerovánu sadu trojúhelníkových oken, kterými vynásobíme získané power spektrum. Trojúhelníková okna jsou vygenerována po spuštění aplikace pomocí funkce *initMelBanks* a jsou uložena ve struktuře *gMelStruc->melBanks*, jejíž definice je uložena v souboru *structures.h.* V této struktuře jsou také uloženy parametry *FFTinL* a *FFTinH*, které definují jaké maximální, resp. minimální, frekvence mají být brány v úvahu při výpočtu koeficientů. Při každém násobení power spektra trojúhelníkovým oknem je vypočítána suma takto upraveného power spektra. Výsledkem je 13 mel-koeficientů. Mel-koeficienty se následně zlogaritmují a na log mel-koeficienty se aplikuje diskrétní kosínova transformace(DCT), čímž obdržíme výsledné mel-frekvenční cepstrální koeficienty. Koeficienty pro výpočet DCT transformace jsou uloženy ve struktuře *gDCTStruc*, která obsahuje předpočítané hodnoty funkce cosinus, která se využívají při výpočtu DCT.

6.4.3 Rozpoznávač

6.4.3.1 Modely slov a jejich uložení

Při rozpoznávání slov musí existovat modely jež bude možné porovnávat s promluvou, ve které chceme rozpoznávat řeč. A tedy modely musí být nějakým způsobem uložené v paměti DSP, aby k nim byl rychlý přístup. Jedním z problémů, oproti implementaci rozpoznávače na PC, je nemožnost využívání modelů, které jsou uloženy na disku počítače, protože DSP jako takové neumožňuje přímý přístup k souborům. Je to logické, neboť u DSP se předpokládá autonomní funkce ve vestavěném systémů, který v naprosté většině případů aplikací DSP nemá uložiště dat k dispozici.

Jedním z řešení je načítání modelů do DSP za chodu počítače pomocí rozhraní RTDX. Tímto způsobem jej bylo v plánu i implementovat. Avšak během implementace se vyskytly problémy, které

to znemožnily. Hlavním problémem je implementace samotné RTDX komunikace, která nefunguje zcela optimálně, protože neumožňuje využívat přerušení, a tedy pro zjištění nově příchozích dat se musí používat metoda tzv. pollingu, což je neustálé dotazování na přítomnost dat. Tento způsob prakticky znemožní jakoukoliv práci, protože procesor je plně vytížen. Dalším problémem je záhadné ztrácení dat při komunikaci. Tyto důvody vedly k tomu, že se od implementace načítání modelů tímto způsobem muselo odstoupit, i když již bylo implementované funkční grafické rozhraní v prostředí Borland C++ Builder.

Jako náhradní způsob řešení, prakticky jediný možný, bylo implementovat modely přímo do zdrojových kódů programů.

Modely slov jsou ukládány do paměti SDRAM, která je k dispozici na vývojové desce ve strukturách, jejichž deklarace je přítomna v souboru s*tructures.h*. Podrobnější popis bude následovat.

Základní strukturou, která vlastně „zaobaluje“ všechny modely slov, je struktura *HMMModles*. V této struktuře je uložen pouze počet modelů a ukazatel na pole ukazatelů na modely.

```
typedef struct{
  Int16 numWords;
  HMMStruc *HMMs;
}HMMModels;
```

Složitější strukturou je již struktura *HMMStruc*, ve které jsou uloženy samotné modely slov.

```
typedef struct{
   char *word;
   unsigned int numParams;
   Int32 frameCount;
   unsigned short numOfStates;
   HMMStateStruc *HMMStates;
   FLOAT_TYPE *HMMTransProbMatrix[3];
   FLOAT_TYPE *tokens;
   Int16      *tokensInd;
   FLOAT_TYPE *precompLike;
 }HMMStruc;
```

V proměnné *word* je uložena znaková podoba slova. Proměnná *numParams* slouží k uložení počtu vstupních parametrů, *frameCount* uchovává počet již zpracovaných rámců, *numOfStates* udává počet stavů Markovova modelu. Následuje ukazatel na pole struktur *HMMStates*, ve kterém jsou uloženy jednotlivé stavy. Každý model musí obsahovat přechodovou matici, definující pravděpodobnosti přechodů mezi jednotlivými stavy, k těmto účelům slouží pole se třemi řádky *HMMTransProbMatrix*, ve kterém jsou uložené log pravděpodobnosti, aby je nebylo nutné počítat během výpočtu. V běžných implementacích se využívá pro ukládání přechodové matice, matice NxN, nicméně uvážíme-li, že užitečná data se nacházejí pouze nad diagonálou, a to ještě pouze maximálně

ve třech diagonálách nad sebou, tak použití matice NxN je zbytečným plýtváním místa. Proto bylo zvoleno tří řádkové pole, ve kterém první řádek obsahuje pravděpodobnost setrvání ve stejném stavu, druhý obsahuje pravděpodobnost přechodu do následujícího stavu a třetí řádek obsahuje pravděpodobnost přeskočení jednoho stavu. Toto pole tedy plně postačuje pro popis přechodové matice HMM modelu.

Dalšími proměnnými, které jsou ve struktuře *HMMStruc* uloženy jsou *tokens*, *tokensInd* a *precompLike*. V poli *tokens* jsou uloženy aktuální hodnoty likelihoodů vypočítané v předchozím kroku Viterbiho algoritmu, *tokensInd* obsahuje indexy stavů, ze kterých přišel aktuální token a *precompLike* obsahuje předpočítané hodnoty vysílacích pravděpodobností v příslušných stavech, tyto hodnoty jsou předpočítávány pokaždé, když se vyhodnocuje nový rámec.

Poslední strukturou, které je potřeba pro popis Markovova modelu je *HMMStateStruc*, ve které jsou uloženy natrénované hodnoty Gaussova rozložení pro příslušný stav.

```
typedef struct{
    Int16 numGauss;
    Int16 numParams;
    FLOAT_TYPE **gaussVariance;
    FLOAT_TYPE **gaussMean;
    FLOAT_TYPE *gConst;
    FLOAT_TYPE *gaussWeight;
}HMMStateStruc;
```

Z názvu proměnných je již možné usuzovat k jakému účelu jednotlivé proměnné slouží. Proměnná *numGauss* obsahuje počet gaussových rozložení, kterými je popsán daný stav, *numParams* obsahuje počet parametrů, v poli *gaussVariance* jsou uložený natrénované hodnoty rozptylů pro patřičný stav, pole *gaussMean* obsahuje střední hodnoty, *gConst* je pole předpočítaných hodnot konstant, které se vyskytují ve vzorci pro výpočet hodnoty gaussova rozložení a konečně *gaussWeigth* je pole vah jednotlivých gaussovek.

Výše popsané struktury plně postačují pro popis skrytých Markovových modelů.

6.4.3.2 Načítání modelů

Jak již bylo uvedeno v předchozí kapitole, tak jsou modely uloženy napevno ve zdrojových kódech, ale aby toto nebylo příliš svazující je možné použít pro vygenerování zdrojového kódu jednoduchý program, který byl pro tyto účely napsán.

Program s příznačným názvem *convertor.exe* umožňuje převod z modelu v textové reprezentaci, ve formátu popsaném níže, do zdrojového kódu použitelného v aplikaci. Jako parametry si z příkazové řádky přebírá názvy souborů s natrénovanými modely. Výstupem tohoto programu je zdrojový kód s názvem *models.h*, který obsahuje několik polí dat, která jsou pak načítána do

připravených datových struktur. Tento model se pomocí *#include* vkládá na patřičnou pozici do funkce *LoadModels* v souboru *dsk_app.c*.

Formát souborů s načítanými daty je v následujícím formátu:

```
Počet stavů HMM modelu
Počet parametrů
Pole středních hodnot pro všechny stavy,
uložené za sebou
[……..]
Pole rozptylů pro všechny stavy uložené za
sebou
[……..]
Matice přechodových pravděpodobností.
[……..]
.
.
[……..]
```

Výsledný soubor se pomocí funkce *LoadModels*, která si jako parametr bere ukazatel na globální strukturu *wordModels*, která je následně naplněna daty ze souboru *models.h*, načte do paměti, při čemž se za asistence makra *mem_alloc,* alokuje dynamická paměť a zároveň se inicializuje.

Pro účely práce jsem využil natrénované modely číslovek, které byly k dispozici v předmětu Zpracování řečových signálů. Modely jsou k dispozici i s referenčními vzorky hlasu, tudíž je možné je dobře testovat. Model každého slova je popsán 16 stavy, každý vektor obsahuje 39 parametrů.

6.4.3.3 Viterbiho algoritmus

Samotné rozpoznávání provádí Viterbiho algoritmus, který pro svou funkci používá výpočet vysílacích pravděpodobností gaussova rozložení pro každý stav skrytého Markovova modelu. Výpočet této hodnoty má na starosti funkce *getDistribution*, která jako parametr vezme strukturu obsahující jeden stav modelu a vypočte výslednou vysílací pravděpodobnost, kterou předá jako návratovou hodnotu funkce.

Jádrem Viterbiho algoritmu je funkce *viterbiEvalModels*, která postupně prochází pole všech modelů a na každý zavolá funkci *viterbiEvaModel*, jež provede jeden krok algoritmu pro právě vyhodnocovaný rámec. Funkce nejprve vypočítá pro všechny stavy modelu vysílací pravděpodobnost, a uloží ji do příslušného pole, ve kterém se uchovávají předpočítané hodnoty. Následně je možné provést jeden krok algoritmu, jenž zahrnuje rozeslání tokenů do navázaných stavů spolu s příslušným přičtením vysílacích a přechodových pravděpodobností.

Pokud dojde poslední rámec, tak dojde k výpočtu výsledných výstupních likelihood, na jejichž základě se rozhodne o nejlepší shodě. Výsledek se následně zobrazí v binární podobě na LED diodách, které jsou integrovány na vývojové desce.

7 Závěr

Cílem diplomové práce bylo navrhnout jednoduchý systém pro rozpoznávání jednotlivých slov na vestavěném zařízení. Původně zamýšlenými prostředky pro implementaci této diplomové práce byla uvažována platforma kombinující FPGA Xillinx Spartan a DSP TMS320C32, jež byla vyvinuta na FIT VUT v Brně pro účely akcelerace grafických algoritmů. Avšak po zralé úvaze jsem došel k závěru, že tato platforma není vhodná pro vývoj rozpoznávače, neboť by tento vývoj byl velice časově náročný, protože při implementaci je nutné řešit zásadní problémy, jako je omezený rozsah čísel, chybějící floating point aritmetika a omezené množství dostupné paměti. Výše uvedené chybějící vlastnosti, by bylo nutné řešit implementací vlastních funkcí, což je úkol velice náročný. Vzhledem k omezeným možnostem uložení samotných modelů, kterých by bylo možno uložit do několika FPGA v počtu jednotek kusů, je přínos diskutabilní, neboť vzhledem k ceně současných FPGA čipů by takový rozpoznávač byl neúměrně drahý. Do budoucna až se možnosti současných FPGA čipů rozšíří je možné uvažovat o jejich nasazení při rozpoznávání řeči. Z výše uvedených důvodu jsem se rozhodl pro náhradní řešení, kterým je implementace jednoduchého rozpoznávače na DSP vývojovém kitu firmy Spectrum Digital, který používá signálový procesor firmy Texas Instruments TMS320C6416.

Rozpoznávač byl implementován na výše uvedeném vývojovém kitu. Při implementaci byl kladen důraz na efektivitu a modularitu, která umožňuje použití i jiných modelů slov než jen těch současně používaných. Implementovaný rozpoznávač používá jako slovník slov české číslovky a díky vysoké míře využití dynamicky alokované paměti, umožňuje v kombinaci s příslušnými utilitami použití dalších modelů pro tyto účely připravených.

Avšak implementace samotná neposkytuje uspokojivé výsledky a bylo by potřeba více času na jeho dokonalé odladění.

Budoucí vývoj implementace spočívá ve vytvoření fungujícího rozhraní, nejlépe grafického, které bude umožňovat načítání složitějších modelů slov, případně fonémů, čímž by bylo dosaženo výrazného nárůstu užitné hodnoty. Dalším vývojovým stádiem by mohlo být implementovat rozpoznávač přímo na vlastní desku, tak aby mohl fungovat jako autonomní systém, který by měl spoustu rozličných využití, např. modul pro řízení domácích spotřebičů.

Oblast rozpoznávání řeči ve vestavěných systémech skýtá veliký prostor pro budoucí rozvoj a tato diplomová práce měla přispět k tomuto rozvoji, což se víceméně povedlo.

Literatura

[1] Gold, B., Morgan, N. *Speech and audio signal processing*. New York, John Wiley & Sons 2000.

[2] Psutka, J., Miller, L., Matoušek, J., Radová, V. *Mluvíme s počítačem česky*. Praha, Academia 2006.

[3] Beauchamp, J. *Analysis, synthesis and perception of musical sounds*. New York, Springer 2007.

[4] Halliday, D., Resnick, R., Walker, J. *Fyzika*, Brno, VUTIUM, 2000.

[5] <*http://encyklopedie.seznam.cz/heslo/474959-fourierova-rada*>, 13. 12. 2007.

[6] Heeger, D., Signals, *Linear Systems And Convolution*, Sep. 26. 2000.

[7] <*http://en.wikipedia.org/wiki/Convolution*>, 10. 12. 2007

[8] *Studijní text Matematika III: Fourierova řady*, UM FSI VUT v Brně

[9] <*http://encyklopedie.seznam.cz/heslo/129030-fourierova-transformace*>, 17. 12. 2007

[10] Slavík, J., *EEG Workshop*, 2003, [online] Dostupné na URL: <*http://www.volny.cz/slavij/DokumentaceHTML/Dokumentace.htm*>

[11] <http://www.dugumkume.org/ses-sentezleme-ve-tanima-teknolojisi-ile-neler-yapilabilir/>, 30. 12. 2007

[12] Černocký, J., *Zpracování řečových signálů – studijní opora*, ÚPGM FIT VUT v Brně

[13] Král, T., *Ročníkový projekt: Rozpoznávač/detektor slov na čipu*, Brno 2006

[14] Šnorek, *Nuronové sítě a nuropočítače*, ČVUT, Praha 2004

[15] <http://www.fs.vsb.cz/books/NeuronoveSite/NN_pojmy.htm>, 1. 1. 2008-01-01

[16] Král, T., *Diplomová práce: Fixed-point implementace rozpoznávače řeči*, Brno 2007

[17] *Hidden Markov Model,*[online] Dostupné na URL<http://en.wikipedia.org/wiki/Hidden_Markov_model>, 2. 11. 2008

[18] Specification of TMS320C32, [online], Dostupné na URL< http://focus.ti.com/docs/prod/folders/print/tms320c32.html>, 3. 1. 2008

[19] *Specification of TMS320C6711*, [online], Dostupné na URL< http://focus.ti.com/lit/ds/sprs088o/sprs088o.pdf>, 3. 1. 2008

[20] *Co je to FPGA/CPLD*, [online], Dostupné na URL< http://gw.asix.cz/zuban/skola-fpga/index2.htm>, 3. 1. 2008

[21] Product specification: Xillinx Virtex, [online], Dostupné na URL < http://www.xilinx.com/support/documentation/data_sheets/ds003-1.pdf>, 3. 1. 2008

[22] Product specification: Xillinx Spartan, [online], Dostupné na URL< http://www.xilinx.com/support/documentation/spartan-3.htm>, 3. 1. 2008

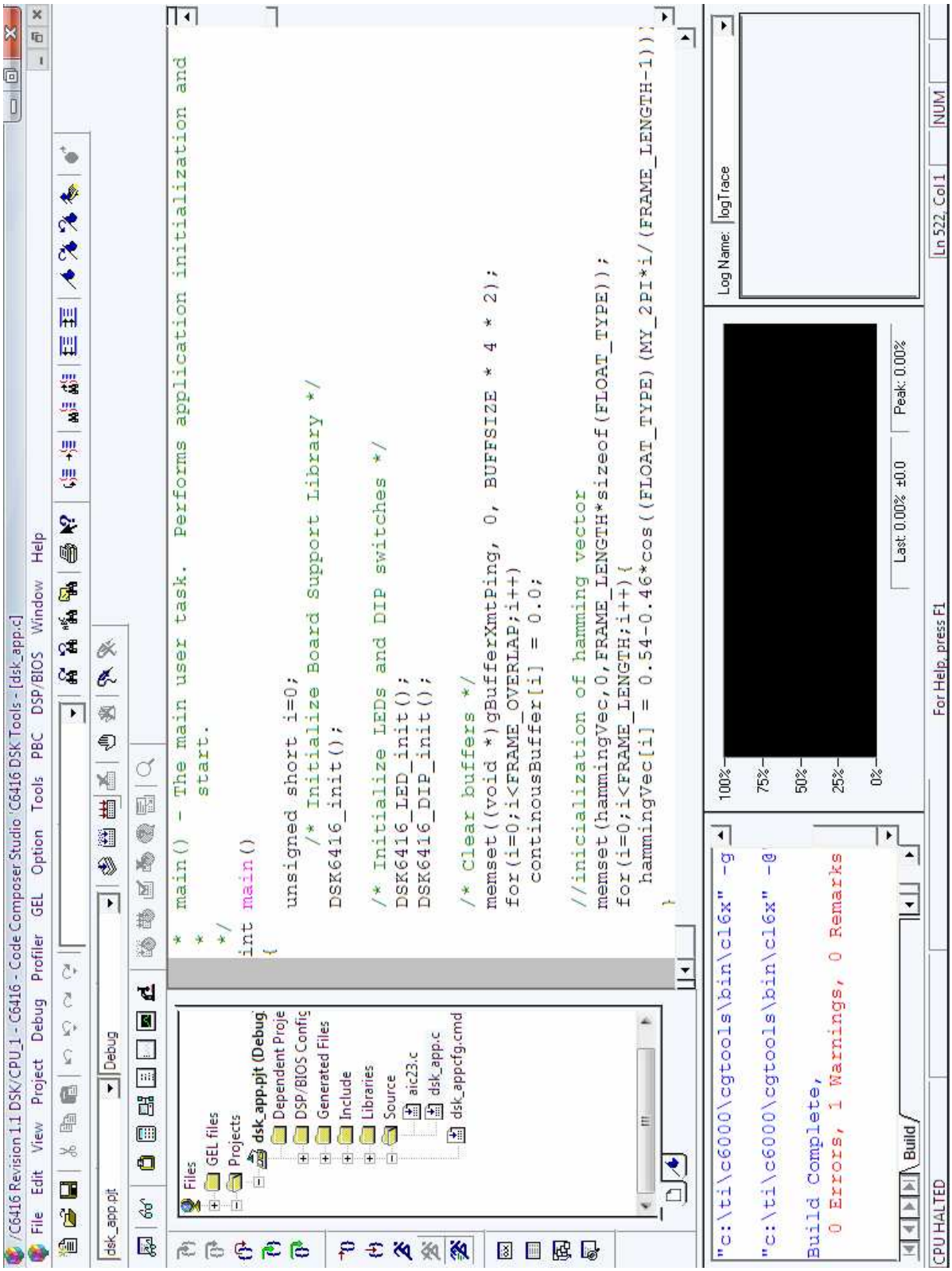
[23] Zemčík, P., Fučík, O., *Obrazové algoritmy s akcelerací hardware,* FIT VUT v Brně

[24] Vejnar, J., *Hopfieldova neuronová síť,* [online] Dostupné na URL <http://neuron.felk.cvut.cz/courseware/data/chapter/36nan062/s32.htm>, 9. 5. 2008

[25] Omondi, A., Rajapakse, J., *FPGA implementations of Neural Networks,* Dordrecht, Springer, 2006.

Seznam příloh

Příloha A – Vzhled aplikace Code Composer Studio



Příloha B – Schéma FPGA/DSP platformy