Vysoké učení technické v Brně

Fakulta informačních technologií

Rozpoznávač / detektor slov na čipu

Ročníkový projekt

Brno, máj 2006 Autor: Tomáš Král

Zadání ročníkového projektu

Rozpoznávač / detektor na čipu

Vedoucí: Burget Lukáš, Ing., Ph.D., UPGM FIT VUT

Zadání:

1. Seznamte se s vhodnou DSP architekturou (ťádová čárka, paměť, programovací jazyky, atd.)
2. Vyřešte online vstup z mikrofonu, pro testování funkčnosti realizujte jenoduchou aplikaci typu „drát“.
3. Seznamte se s toolkitem STK a s nástroji pro parametrizaci, rozpoznávání řeči a detekci klíčových slov, otestujte je s dodanými modely na PC.
4. Adaptujte vybrané nástroje či jejich součásti na DSP, ověřte jejich funkčnost na jadnoduché úloze.

Prehlásenie

Prehlasujem, že tento ročníkový projekt som vypracoval samostatne pod vedením Ing. Burgeta Lukáše, Ph.D. a Doc. Dr. Ing. Černockého Jana.

Všetky zdroje z ktorých som čerpal informácie, ako knižné tituly, výukové materiály a iné dokumenty, som uviedol v záverečnej časti dokumentu.

V Brne, dňa 10.mája 2006 …….....

Tomáš Král

Abstrakt

Tento ročníkový projekt sa zaoberá problematikou automatického rozpoznávania reči. Cieľom projektu, ktorý je na týchto stránkach dokumentovaný, bolo v prvom rade zoznámiť sa so systémami automatického rozpoznávania reči a následne navrhnúť riešenie, akým by bolo možné systém rozpoznávania reči zabudovať do embedded systémov. Bolo treba zanalyzovať všetky metódy a vlastnosti dnešných rozpoznávačov, ktoré by potencionálne mohli byť portované na takéto systémy a kandidáta s najlepšími predpokladmi splniť zadanú úlohu, implementovať na vhodnú hardwarovú architektúru.

Kľúčové slová

Speech recognition systems, Feature Extraction, Hidden Markov Models, embedded systems, DSP, Code Composer Studio

Obsah:

Predslov

Systém rozpoznávania reči, typu Key Word Spotting, adaptovaný na embedded system: toto si dáva za cieľ projekt zadaný Ústavom počítačovej grafiky a multimédií, na ktorom som pracoval.

Tento dokument je koncipovaný tak, aby nás postupne previedol cez jednotlivé fázy vývoja tohto systému. **1. kapitola** nás uvedie do širšieho kontextu problému rozpoznávania reči a načrtne zmysel projektu (rozpoznávanie na čipe) V **2.kapitole** sa pozrieme na teóriu, ktorá sa týka rozpoznávania a uvedieme si vždy niekoľko alternatív riešenia toho-ktorého podproblému. Čo sú to embedded systémy, ich vlastnosti a informácie o konkrétnej hardwarovej architektúre, to sú témy **3.kapitoly**. Vlastné riešenie portovania rozpoznávača reči na embedded system a jeho testovanie nájdeme v **4.kapitole**, resp. **5.kapitole** a záverečná **6.kapitola** zhrnie dosiahnuté výsledky a ďalšie potrebné kroky k dosiahnutiu ideálneho rozpoznávania na zvolenom čipe.

Poďakovanie

Na tomto mieste by som chcel vyjadriť poďakovanie Ing. Burgetovi Lukášovi a Doc.Dr.Ing. Černockému Janu, ktorý mi poskytli nezbytné informácie a rady pri riešení tohto projektu a bez ktorých pomoci by dokončenie práce trvalo oveľa dlhšie. Tak isto by som chcel poďakovať aj Ing. Schwarzovi Petrovi za poskytnuté konzultácie.

1. Úvod

1.1 Rozpoznávanie reči

Systémy pre rozpoznávanie reči sú v dnešnej dobe veľmi aktuálnou, mnohými odborníkmi diskutovanou a vo veľa výskumných laboratóriách skúmanou oblasťou, ktorá do dnešnej doby priniesla tak ako určité pokroky a úspechy, aj neúspechy a stále nové a nové otázky na ktoré potrebujeme nájsť odpoveď tak, aby bol zachovaný určitý technologický rast a pokrok. To, že po technológiách týkajúcich sa systémov rozpoznávania reči je obrovský dopyt je nepopieratelné. Výrobcovia spotrebnej elektroniky by určite radi poskytli spotrebiteľovi možnosť prijať hovor na mobilnom telefóne príkazom „accept“, či pohodlné zníženie hlasitosti výkonnej audio Hi-Fi sústavy vyslovením príkazu „volume down“. Posunutím o stupienok vyššie v technológií rozpoznávania reči by sme sa mohli dostať do automobilového priemyslu. Tu už okrem štandartných príkazov ktoré poznáme z ovládania spotrebnej elektroniky požadujeme aj príkazy ako „find hotel“, ktorý aktivuje systém GSP, alebo príkaz „report speed limit“, ktorý v kooperácií s ostatnými systémami informuje o povolenej rýchlosti v danom mieste.

Tieto príklady predstavujú základnú myšlienku využitia systémov rozpoznávania reči v praxi. Vzniká teda otázka, ako ďaleko sme sa v tomto smere dostali. Všetko závisí od kvality rozpoznávania ktorú požadujeme. Jednoduché rozpoznávače ako poznáme z mobilných telefónov a iných malých elektronických zariadení sú založené na jednoduchom porovnávaní rozpoznávaného vzorku a referenčného vzorku, ktorý bol v minulosti do telefónu nahraný uživateľom. Zariadenie teda reaguje len na hlas jedného užívateľa od ktorého sa očakáva, že príkaz zopakuje rovnakým tónom a rovnakou rýchlosťou. Takýmto systémom chýba akákoľvek robustnosť. Princíp technológie o niečo dokonalejších rozpoznávačov pri ktorých nezáleží či rozpoznávané slovo povedalo dieťa so základným tónom na vysokých frekvenciách, alebo muž so základným tónom na nízkych frekvenciách, prípadne že rozpoznávané slovo bolo povedané rozdielnou rýchlosťou, je tiež zvládnutý. Avšak s technologickou dokonalosťou sa úmerne zvyšujú aj hardwarové nároky. Tu už narážame na problém aplikácie takýchto vyspelejších systémov napríklad do mobilných telefónov, ktorých hardwarové vybavenie nieje dostatočné. Ďalším problémom, ktorý je za potreby v každom prípade nejakým spôsobom minimalizovať je vplyv šumu na rozpoznávanie reči. Aj rozpoznávače pochádzajúce z renomovaných výskumných ústavov majú veľké problémy so šumom, ktorý je prítomný všade okolo nás. Šoférom by sa napríklad určite nepáčilo, ak by ich auto prestalo reagovať na hlasové pokyny zakaždým keď okolo prejde hlučný kamión.

Až po vyriešení týchto, svojím spôsobom elementárnych problémov sa v budúcnosti budeme môcť začať zaoberať vysoko sofistikovanými systémami pre rozpoznávanie reči. V prvom rade sa jedná o systém, ktorý bude schopný z hovoreného slova vytvárať textové dokumenty. Ďalej systémy, ktoré budú splňovať požiadavky národných bezpečnostných inštitúcií. Jedná sa najmä o rozpoznávače, ktoré by mohli byť nasadené na rizikových miestach ako letiská, vlakové stanice, či veľké obchodné centrá a boli by schopné poradiť si so šumom, ktorý dav ľudí spôsobuje a filtrovať len dôležité, resp. podozrivé signály.

1.3 Rozpoznávanie na čipe

Rozpoznávače reči sú z pochopiteľných dôvodov najefektívnejšie na osobných počítačoch, ktoré im poskytujú dostatočné hardwarové i softwarové prostriedky. Bohužiaľ v praxi, na miestach kde sa bežne stretávame, alebo zatiaľ by sme sa len chceli stretávať s kvalitnými rozpoznávačmi väčšinou nieje priestor pre fyzicky veľké osobné počítače.

Z tohto dôvodu nemenej dôležitou úlohou pri vývoji systémov pre rozpoznávanie reči je aj ich optimalizácia do takej podoby, aby boli schopné fungovania aj na systémoch s obmedzenými hardwarovými či softwarovými prostriedkami, na tzv. embedded systémoch.

Cieľom tohto projektu je implementovať rozpoznávač reči na čip. Pri hľadaní vhodného čipu, ktorý by sa na túto úlohu najlepšie hodil nebolo treba dlho premýšľať. Voľba padla na DSP. Digitálne signálové procesory poskytujú prostriedky, ktoré sú pre funkčnosť rozpoznávača reči nevyhnutne potrebné. Disponujú dostatočným výpočetným výkonom, sú pripojiteľné k externým zariadeniam a prostriedkom ako je pamäť, audio codec, paralelné, sériové a iné rozhrania.

2. Teória rozpoznávania

2.1 Rozpoznávanie reči a veda

Problematika počítačového rozpoznávania reči je stará takmer ako počítače samotné. Vychádza to z faktu, že najprirodzenejšou formou komunikácie pre človeka je verbálna forma. Už od samotných počiatkov vedného oboru Computer Science, vedci začali premýšľať o možnosti naučiť počítače porozumieť tejto forme komunikácie. Avšak postupom času vedci prišli na to, že táto, na prvý pohľad možno nie až tak komplikovaná úloha, v sebe ukrýva veľké množstvo problémov, ktoré je potrebné vyriešiť. Pozrime sa teda pre názornosť, vedomosti ktorých vedných oborov potrebujeme pri vývoji systémov rozpoznávania reči:

- spracovanie signálov - z rečového signálu potrebujeme získať relevantné informácie
- fyzika: akustika - porozumenie fyziologických mechanizmov, ktorý produkujú a prijímajú rečové signály
- pattern recognition - algoritmy klastrujúce dátové vzorky a produkujúce prototypy reprezentujúce jednotlivé klustery
- komunikácia a teória informácie - procedúry na odhadovanie parametrov, detekciu prítomnosti určitého rečového vzorku, …
- fonetika - vzťahy medzi zvukmi
- syntaxa, sémantika - gramatika a význam slov
- informatika - vývoj efektívnych algoritmov
- a mnoho ďalších...

Tento krátky výčet vedných oborov človeka presvedčí, že problematika systémov rozpoznávania reči nieje jednoduchá záležitosť.

2.2 Prístupy k rozpoznávaniu reči

Dnešné rozpoznávače reči môžeme rozdeliť do rôznych kategórií podľa toho na akom princípe fungujú ich jadrá. Vo všeobecnosti existujú tri prístupy k rozpoznávaniu reči:

- akusticko-fonetický prístup
- tzv. pattern-recognition prístup
- prístup založený na umelej inteligencii

Akusticko-fonetický prístup je založený na rozpoznávaní jednotlivých fonémov a ich vlastností, ktoré sa vyskytujú v tom ktorom jazyku. Vlastnosti fonémov sú však veľmi premenné nielen v závislosti na rečníkovi, ale aj v závislosti na susedných fonémoch. To znamená, že jeden a ten istý foném môže mať, resp. aj má, rôzne vlastnosti ak je v spojení s dvoma rôznymi inými fonémami. Prístup založený na tomto princípe v sebe skrýva mnoho problémov a ťažkostí, ktoré sa obtiažne riešia.

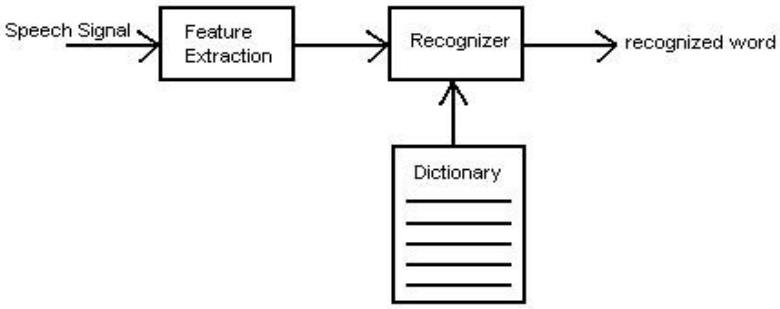
Pattern recognition – tento prístup pozostáva z dvoch krokov. V prvom rade je to trénovanie. Trénovacie vzorky systém klasifikuje a naučí sa ktoré akustické vlastnosti vzoriek sú relevantné a spoľahlivo reprezentujú trénovaný vzor. V druhom kroku po natrénovaní sa v samotnom rozpoznávaní porovnáva vzor na vstupe s referenčnými vzorkami. Táto metóda má niekoľko výhod oproti akusticko- fonetickému prístupu. Je jednoduchšia, robustnejšia a poskytuje lepšie výsledky.

Posledný prístup založený na umelej inteligencii je akousi kombináciou oboch predošlých prístupov, pretože používa metódy a koncepty, ktoré nájdeme ako v akusticko-fonetickom prístupe tak aj v pattern recognition.

Po zvážení všetkých výhod a nevýhod jednotlivých prístupov sa ako najvhodnejšia metóda pre aplikáciu rozpoznávača na čipe javí metóda pattern recognition. Keďže na čipe, resp. embedded systémoch máme k dispozícii obmedzené prostriedky, jednoduchosť, efektívnosť a v neposlednom rade aj robustnosť rozhodla práve v prospech pattern recognition. V nasledujúcom texte teda budeme brať do úvahy už len Pattern recognition.

2.3 Dekompozícia problému rozpoznávania

Systém rozpoznávania reči je proces, ktorý je značne komplikovaný a pre pochopenie jeho činnosti je potrebná dekompozícia jednotlivých častí do logických blokov.



Rečový signál, ktorý je produkovaný hlasovým ústrojenstvom na jednej strane a prijímaný posluchovým ústrojenstvom na druhej , v sebe nesie obrovský informačný obsah, ktorý je ľudský mozog schopný bez problémov spracovať a extrahovať si potrebné informácie. Bohužiaľ dnešné počítače ešte nedisponujú takou výpočetnou silou, ktorá by tohto bola schopná. Z tohto dôvodu potrebujeme na začiatku rečový signál upraviť do takej formy, aby bol použiteľný v počítačovom spracovaní. Okrem relevantných informácií sú v rečovom signále prenášané aj informácie o človeku, ktorý rozpráva, farbe jeho hlasu, nálade, prostredí, … Tieto informácie sú pre systémy

rozpoznávania reči nadbytočné a preto ich môžeme odstrániť. K takejto úprave rečového signálu je určená parametrizácia, tzv. Feature Extraction.

Po získaní parametrov reči nasleduje samotné rozpoznávanie založené na princípe porovnávania vstupného vzorku s referenčnými vzorkami. V tomto momente vzniká ďalší problém. Vzorky, resp. slová, ktoré rozpoznávame totiž vo väčšine prípadov niesu rovnako dlhé. Riešením by mohla byť lineárna transformácia a vyrovnanie dĺžky porovnávaných vektorov, avšak v praxi sa ukázalo, že toto nieje ideálne riešenie. Pri skracovaní vektoru totiž v zásade musíme niektoré prvky odignorovať a nikdy nevieme či nevylúčime práve dôležitú informáciu. Alebo môžeme vektor rozšíriť do požadovanej dĺžky, avšak v tomto prípade zase musíme zaviesť nové prvky, ktoré môžu tak isto negatívne ovplyvniť výsledok. Ideálnym riešením je tzv. dynamické programovanie. Dynamic Time Wrapping (DTW) a Hidden Markov Models (HMM) sú techniky dynamického programovania, ktoré dokážu porovnávať vzorky rozdielnej dĺžky a vrátiť nám mieru podobnosti dvoch porovnávaných vektorov, resp. rozpoznať slovo.

Ako už bolo spomenuté pri pattern recognition rozpoznávaní reči je nepostrádateľnou súčasťou systému aj trénovanie na množine trénovacích vektorov, slov, ktoré chceme rozpoznávať. V prípade HMM bude natrénované slovo reprezentované množinou viacrozmerných (typicky 13, alebo 39 rozmerných) Gaussových rozložení a pravdepodobnosťami prechodov medzi jednotlivými stavmi HMM. Pri DTW, najjednoduchším spôsobom je jedna referenčná sekvencia vektorov reprezentujúca rozpoznávané slovo. V ideálnom prípade je to viac referenčných sekvencí pre slovo, alebo vytvorenie priemerného vzoru z viacerých referenčných sekvencí pre každé rozpoznávané slovo.

2.4 Feature Extraction

Z vedných oborov, ktoré boli spomenuté, že sa významnou mierou podieľajú na rozpoznávaní reči, by sme mohli spracovanie signálov postaviť do pozície akéhosi základu, motora celého procesu rozpoznávania. Tento fakt je daný tým, že rečový signál musíme nejakým spôsobom parametrizovať tak, aby boli tieto parametre prijateľné a spracovateľné výpočtovými systémami. Vlastnosťou parametrov, ktorú sa zakaždým snažíme dosiahnuť je to, aby parametre zachytávali minimálnu množinu tých vlastností rečového signálu, ktoré sú pre zvolený systém potrebné. Ostatné informácie a vlastnosti by mali zostať pre systém neznáme a tým ho nezaťažovať.

V zásade rozlišujeme dva spôsoby popisu signálu. Neparametrický, ktorý je založený len na znalosti spracovania signálov(banky filtrov, Fourierova transformácia,...) Pri parametrickom popise sú vyžadované aj znalosti o tvorbe reči. V praxi sa avšak väčšinou používa kombinácia týchto dvoch spôsobov popisu.

Ako teda parametre signálu vznikajú? Jednotlivé parametre rečového signálu sú vlastne vektory čísiel, ktoré zachytávajú dôležité vlastnosti krátkeho časového úseku, tzv. rámca, signálu. Aby sme boli schopný zachytiť všetky dôležité vlastnosti a zmeny signálu, potrebujeme pracovať nad segmentmi, rámcami signálu špecifickej dĺžky. Na signál, ktorý považujeme za vo všeobecnosti za náhodný, sa potrebujeme v rámci pozerať ako na stacionárny. Rámec musí byť teda dostatočne krátky, ale na druhú stranu aj dostatočne dlhý na to, aby sme boli schopný zachytiť požadované relevantné vlastnosti. Rámce musia byť ďalej navzájom prekryté, aby sme zachytili každú zmenu. Hodnoty, ktoré sa ukázali v praxi ako vyhovujúce sú nasledovné: v prvom rade vzorkovacia frekvencia rečového signálu 8000Hz je postačujúca. Dĺžka jednotlivých rámcov sa pohybuje v rozmedzí 20-25ms (tj. 160-200 vzoriek pri 8kHz) a prekrytie rámcov sa nastavuje na hodnotu 10ms. Použitím týchto hodnôt získame pre signál dlhý jednu sekundu až 100 rámcov, resp. parametrov.

Posledným krokom v tzv. predpríprave na parametrizáciu je použitie window function. Typicky sa používa Hammingovo okno, ktoré svojím spôsobom utlmí význam vzoriek, ktoré sa nachádzajú pri na okrajoch jednotlivých rámcov a vyzdvihne význam hodnôt v okolí stredu rámca.

Až na tomto mieste sa začína parametrizácia rečového signálu. Existuje široké spektrum možností parametrického vyjadrenia rečového signálu. Skalárne parametre (jedna hodnota pre jeden rámec) počítané na základe strednej krátkodobej energii dokážu detekovať rečovú aktivitu, rozlíšiť znelé a neznelé hlásky. Sú však veľmi náchylné na šum. Počet priechodov nulou, počet priechodov úrovňou a iné skalárne parametre sa tak isto obmedzujú len na špecifické použitie, ktoré sú pre rozpoznávanie reči nevyhovujúce. Potrebujeme teda vektorové parametre, ktorých informačná hodnota je oveľa väčšia. Spektrálna analýza sa ukazuje ako veľmi silný nástroj pre popis parametrov rečového signálu. Medzi dve najrozšírenejšie metódy spektrálnej analýzy patria: spektrálna analýza založená na banke filtrov a spektrálna analýza založená na Linear Predictive Coding (LPC).

Banky filtrov

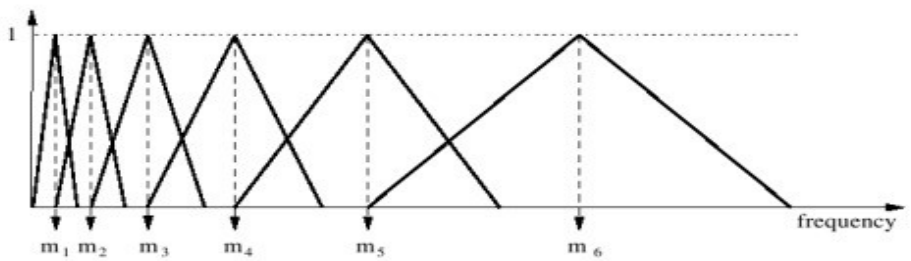
Metoda spočíva v tom, že vstupný signál sa prefiltruje filtrami pásmovej priepustnosti, ktoré sa rozprestierajú a prelínajú na tej časti spektra, ktorá nás zaujíma. Výstupom každého filtra je tzv. krátkodobé spektrum.

Cepstrálne koeficienty: ako už bolo spomenuté, rečový signál má obrovskú informačnú hodnotu, ktorú dnešné počítače nedokážu v reálnom čase spracovávať. V prvom rade sa teda snažíme odstrániť nepotrebné informácie o budení signálu (t.j. šum a základný tón reči, funkcia hrtanu a hlasiviek) a zaujímajú nás len informácie o modifikácii signálu(t.j. funkcia hlasového, artikulačného traktu) Rečový signál je vlastne konvolúciou budenia a impulznej odozvy filtra, ktorý je tvorený artikulačným traktom. Našou úlohou teda je tieto dve zložku od seba oddeliť a zdroj budenia do ďalšieho spracovania už nezaradiť. DFT-cepstrum dokáže túto konvolúciu rozdeliť.

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

Výsledné cepstrálne koeficienty c(n), sú už kvalitatívne na vyhovujúcej úrovni na to, aby mohli byť použité pri rozpoznávaní reči.

Mel-frekvenčné cepstrálne koeficienty (MFCC): Avšak až zavedením Mel-frekvenčných cepstrálnych koeficientov získavame relatívne ideálne koeficienty, ktoré sa v praxi využívajú. Zlepšenie oproti klasickým cepstrálnym koeficientom vychádza z faktu, že DFT má na celom spektre rovnaké frekvenčné rozlíšenie, na rozdiel od ľudské ucha, ktorého rozlišovacia schopnosť klesá s narastajúcou frekvenciou. Z tohto dôvodu umiestnime na frekvenčnú osu nelineárne filtre(tzv. Mel banky, ktorých je 23), zmeriame energie na ich výstupoch a použijeme ich pre výpočet cepstra (namiesto DFT)



Výpočet koeficientov na jednom rámci prevedieme nasledovne: z 200 vzoriek v jednom rámci získame pomocou Fourierovej transformácie frekvenčné spektrum, ktoré bude mať 256 hodnôt a do ďalšieho spracovania vezmeme len prvých 128 hodnôt (ďalších 128 sú tie isté hodnoty len zrkadlovo otočené) Spektrum umocníme a aplikujeme 23 Mel-baniek (tj. vynásobenie trojuholníkovým oknom a sčítanie). Na 23 hodnôt, ktoré týmto spôsobom získame, aplikujeme logaritmus a spätnú Fourierovu transformáciu. V skutočnosti sa ale nejedná o klasickú spätnú Fourierovu transformáciu, ale realizujeme diskrétnu kosínusovú transformáciu, z ktorej nám vypadne 13 čísiel. Získali sme teda 13 Mel-frekvenčných cepstrálnych koeficientov (MFCC). V praxi sa z týchto 13 koeficientov, ešte transformáciami získa ďalších 23 hodnôt, čo nám dáva dokopy typický počet, 39 MFCC. V projekte som sa kvôli hardwarovým prostriedkom, ktorých na DSP nieje veľa, obmedzil na pôvodný počet 13 MFCC. Pre potreby rozpoznávania reči je 13 koeficientov postačujúcich.

Linear Predictive Coding

Druhom metódou spektrálnej analýzy je LPC. Táto metóda poskytuje výborný model rečového signálu, je matematicky presne definovaná , relatívne jednoduchá a ľahko sa implementuje jak softwarovo, tak i hardwarovo. Výpočtový výkon potrebný pre LPC je výrazne menší ako pri banke filtrov. Systémy rozpoznávania reči založené na LPC podávajú v praxi výborné výsledky, v niektorých prípadoch dokonca lepšie ako rozpoznávače založené na banke filtrov. Princíp fungovania LPC spočíva v tom, že daná vzorka signálu v čase t, s(t), môže byť aproximovaná ako lineárna kombinácia n predošlých rečových vzoriek.

$$s(n) = a_1 s(n-1) + a_2 s(n-2) + \ldots + a_p s(n-p) ,$$

kde koeficienty $a_1, a_2, \ldots, a_p$ sú neznáme a je potrebné ich odhadnúť. Ďalšie teoretické informácie o LPC možno nájsť v odkazovanej literatúre.

2.5 Recognition

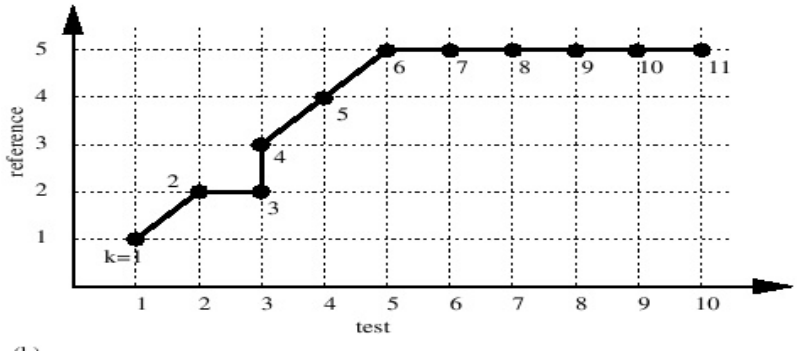
V tomto momente teda máme k dispozícii nástroje ako MFCC a LPC, ktoré nám parametricky popisujú rečový signál a môžeme začať so samotným rozpoznávaním. Systémy rozpoznávania reči môžeme rozdeliť podľa funkcie na:

- rozpoznávače izolovaných slov
- rozpoznávače spojených slov z obmedzeného slovníka
- rozpoznávače plynulej reči s veľkým slovníkom

V nasledujúcom výklade sa obmedzíme len na rozpoznávanie izolovaných slov. Majme teda slovo X, ktoré chceme rozpoznať. Toto slovo bude po parametrizácií reprezentované sekvenciou vektorov O = [o(1),o(2),o(3), … o(T)]. Na druhej strane, v slovníku máme uložené natrénované sekvencie vektorov, ktoré popisujú rozpoznávané slová. Našou úlohou je určiť, či sa slovo, ktoré je privedené na vstup nachádza v slovníku a prípadne ho identifikovať. Na výber máme možnosť rozpoznávania na základe určenia vzdialenosti medzi dvoma vektormi, alebo rozpoznávanie na základe štatistického modelovania. Pre zopakovanie treba pripomenúť, že dĺžky jednotlivých sekvencií niesu rovnaké a preto musíme použiť metódy dynamického programovania.

Dynamic Time Wrapping

DTW je metóda dynamického programovania, ktorá počíta vzdialenosti medzi rôzne dlhými sekvenciami vektorov, referenčných a testovacích. Pre výpočet vzdialenosti vektorov budeme potrebovať dve transformačné funkcie r(k) a t(k) , ktoré získame z grafu porovnávania sekvencie vektorov.



Cesta C v tomto grafe je daná dĺžkou K a priebehom funkcií r(k) a t(k). Potom vzdialenosť sekvencí O a R pre túto cestu je daná vzorcom:

$$D_C(\mathbf{O},\mathbf{R}) = \frac{\sum_{k=1}^{K_C} d[\mathbf{o}(t_C(k)),\mathbf{r}(r_C(k))]W_C(k)}{N_C}$$

a výsledná vzdialenosť cez všetky možné cesty je definovaná ako:

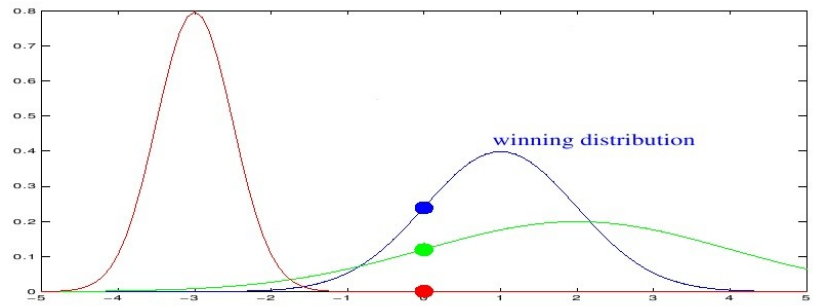
$$D(\mathbf{O},\mathbf{R}) = \min_{\{C\}} D_C(\mathbf{O},\mathbf{R}).$$

Rozpoznávač vypočíta vzdialenosti testovacej sekvencie cez všetky referenčné vzorky a najmenšou vzdialenosťou identifikuje, rozpozná slovo. Ďalšie informácie o DTW možno nájsť možno nájsť v odkazovanej literatúre.

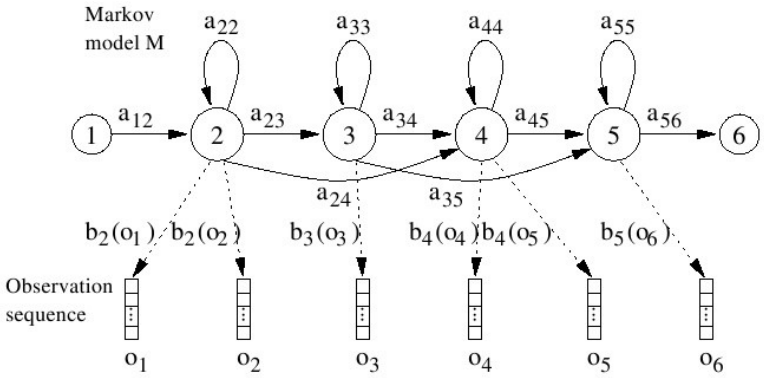
Hidden Markov Models

Metóda HMM bola použitá pri riešení tohto projektu, takže sa na ňu pozrieme trocha podrobnejšie. Na rozdiel od DTW, ktorá počíta vzdialenosti medzi jednotlivými vektormi a my sa snažíme nájsť najmenšiu vzdialenosť, HMM využíva štatistické modelovanie, ktoré spočíva získaní najväčšej možnej pravdepodobnosti, že zadaný model generuje požadovanú sekvenciu vektorov. Čo to znamená, si teraz vysvetlíme.

V prvom rade treba pochopiť akou formou sú natrénované slová, ktoré chceme rozpoznávať uložené v slovníku. Tou formou je Gaussovské rozloženie pravdepodobnosti. Ak by slová boli reprezentované jedným skalárom, po natrénovaní 3 slov by Gaussovské rozloženia mohli vypadať nasledovne:



Avšak pochopiteľne, slová nemôžu byť reprezentované len jedným skalárom. V prípade popisu slova jedným vektorom by sme dostali n-rozmerné Gaussovské rozloženie. Avšak my už vieme, že jednotlivé slová sú reprezentované celou sekvenciou, 13 alebo 39 rozmerných vektorov. Logicky sa ponúka riešenie, vytvoriť pre každý vektor jedno Gaussovské rozloženie. Pripomeňme si však, že slová majú väčšinou rozdielnu dĺžku. Zavedieme teda modely, v ktorých sa jednotlivé Gaussovské rozloženia môžu opakovať. Týmto krokom v skutočnosti vytvoríme skryté Markovove modely HMM. Takže jednotlivé slová v slovníku budú popísané Markovými modelmi, v ktorých budú logicky umiestnené a poprepájané Gaussovské rozloženia pravdepodobnosti.

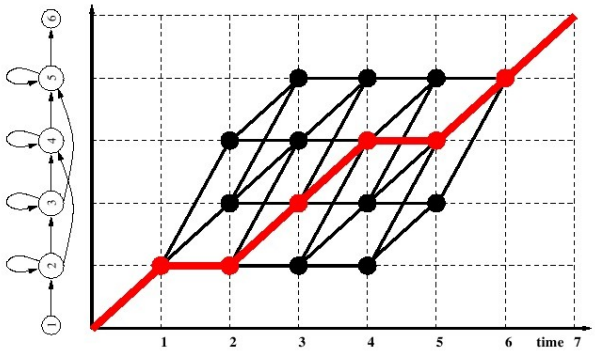


Jednotlivé funkcie hustoty rozloženia vysielacích pravdepodobností pre P-rozmerné Gaussovské rozloženie sa počítajú:

$$b_j[\mathbf{o}(t)] = \mathcal{N}(\mathbf{o}(t); \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) = \frac{1}{\sqrt{(2\pi)^P |\boldsymbol{\Sigma}_j|}} e^{-\frac{1}{2}(\mathbf{o}(t)-\boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{o}(t)-\boldsymbol{\mu}_j)},$$

Teraz keď už vieme počítať vysielacie pravdepodobnosti Gaussovských rozložení (= stavy HMM) môžeme vypočítať celkovú pravdepodobnosť, že model M generuje sekvenciu vektorov O. Jednotlivé vektory sekvencie O, sa pokúsime rozložiť na HMM stavy tak, aby sme získali najväčšiu

možnú pravdepodobnosť. V nasledujúcom obrázku sú zobrazené všetky možné prechody v HMM,



… a červenou farbou je vyznačená postupnosť prechodov X=[1,2,2,3,4,4,5,6], ktorá nám dáva (teoreticky) najväčšiu pravdepodobnosť generovania sekvencie O modelom M. Pri pohľade na tento graf je hneď jasné, že všetkých možných postupností prechodov je veľmi veľa a výpočet pravdepodobností pre všetky prechody by bol veľmi náročný. Pri riešení tohto problému nám našťastie pomôže efektívny Viterbiho algoritmus. Ten si priebežne uchováva najlepšie dočasné výsledky a na konci v čase t keď sa dostane k poslednému prvku vektora O(t) je okamžite schopný určiť najväčšiu pravdepodobnosť generovania sekvencie O modelom M.

Rozpoznávač reči fungujúci na základe HMM teda funguje nasledovne. V prvom rade si natrénujeme modely HMM pre slová ktoré chceme rozpoznávať. Na vstup nám príde sekvencia parametrov slova O. Sekvenciu aplikujeme na všetky modely v slovníku za pomoci Viterbiho algoritmu a ten nám postupne vráti pravdepodobnosti generovania O jednotlivými modelmi. Model s najväčšou pravdepodobnosťou predstavuje rozpoznané slovo.

3. Rozpoznávanie reči na DSP

3.1 DSP

Ako už bolo povedané v úvodnej kapitole, v praxi pri využití menej či viac dokonalých a inteligentných rozpoznávačov reči sa kladie veľký dôraz na to, aby tieto systémy boli použiteľné na takých výpočtových architektúrach, ktoré majú väčšinou do určitej miery obmedzené, jak hardwarové tak i softwarové prostriedky. Hovoríme teda o takzvaných embedded systémoch. Do skupiny embedded systémov môžeme teda zaradiť väčšinu spotrebnej elektroniky, ktorá je riadená rôznymi mikroprocesormi, mikrokontrolérmi či inými riadiacimi výpočtovými jednotkami. Digitálne signálové procesory nevynímajúc.

Procesory ktoré sú navrhované na digitálne spracovanie signálov (DSP) sa vyznačujú určitými vlastnosťami, ktoré na jednej strane ostatné procesory postrádajú, ale naopak taktiež nedisponujú niektorými vlastnosťami, ktoré sú samozrejmosťou na iných HW architektúrach. Preto je vždy dôležité zodpovedať si na základné otázky týkajúce sa požadovaného výkonu, HW a SW prostriedkov, fyzických vlastností , a v neposlednom rade aj otázky ohľadne obstarávacích a prevádzkových nákladov. Po zvážení všetkých týchto faktorov by sme mali vybrať vhodnú hardwarovú architektúru pre náš systém.

Podobnými úvahami som sa dopracoval k odpovedi na otázku, aký čip použiť na implementáciu systému pre rozpoznávanie reči. Digitálny signálový procesor splňuje všetky požiadavky systému, ktorý by mal byť schopný poskytnúť potrebné zdroje pre rozpoznávač reči. Výpočtový výkon je síce o niečo menší ako poskytujú iné procesorové jednotky, avšak keďže je sústredený len na výpočty týkajúce sa rozpoznávania, je tento výkon dostačujúci. Po pripojení externej pamäte RAM odpadá aj problém nedostatku pamäti. Avšak azda najväčšou výhodou tohto riešenia sú rozmery takto koncipovaného systému, ktoré by nemali presiahnuť rozmery napríklad bežne používaného mobilného telefónu.

Systém rozpoznávania reči bol v tomto projekte implementovaný na vývojový KIT DSK TMS320C6416 osadený DSP procesorom C6416 od firmy Texas Instruments. Vývojový Kit obsahuje všetky potrebné prostriedky ako pamäť, audio codec, rozhranie k pripojeniu k PC a iné.

3.2 Architektúra TMS320C64x

DSP procesory z rady TMS320 sa delia na fix-point, floating-point a multiprocesorové signálové procesory. Ich architektúra je navrhnutá špeciálne pre real-timeové signálové spracovanie.

TMS320 procesory:

- floating-point DSPs: C3x, C4x
- multiprocessor DSPs: C8x
- fix-point DSPs: C1x, C2x, C5x, C6x

Ako už z názvu vyplýva, KIT ktorý bol použitý pre rozpoznávač reči je osadený procesorom z rady C64x. Tento procesor patrí do skupiny fix-pointových procesorov. Obsahuje 64, voľne použiteľných, všeobecných 32-bitových registrov, dvojúrovňovú cache pamäť (32KB L1, 1024KB L2), 8 funkčných jednotiek: 2 násobičky, 6 aritmeticko-logických jednotiek. Avšak i napriek tomu, že CPU postráda floating-point výpočtové jednotky, táto architektúra sa s floating-point číslami dokáže vysporiadať a to tak, že floating-point operácie sa emulujú. Samozrejme, že efektívnosť výpočtov týmto pádom klesá, avšak pri neveľkom využívaní tejto emulácie je pokles výkonnosti zanedbateľný. Tento VLIW CPU je schopný vykonávať až 8 32-bitových inštrukcií v jednom cykle. Podporuje 8/16/32 bitové dátové typy čo zvyšuje efektívnosť využitia pamäti. 40-bitové aritmetické operácie zaručujú väčšiu presnosť výpočtov. 32-bitový adresový priestor dovoľuje adresovať internú pamäť, ktorá má oddelený programový a dátový priestor. Pri použití externej pamäte je adresový priestor zjednotený spolu s internou pamäťou za pomoci External Memory InterFace (EMIF). EMIF podporuje externé SDRAM, SBSRAM, SRAM. K dispozícii je ďalej DMA radič, ktorý má 4 programovateľné kanály. Funkcionalitu DMA radiča poskytuje aj dokonalejší EDMA radič, ktorý má dokopy 16 kanálov a aj pamäťový priestor pre uloženie rôznych konfigurácií DMA prenosov pre neskoršie použitie. Komunikácia s okolím prebieha cez paralelný port HPI, alebo multikanálový sériový port McBSP. Vo výbave nájdeme aj 3 32-bitové časovače, ktoré slúžia napr. na časovanie udalostí, generovanie pulzov, generovanie prerušení, či synchronizačné účely DMA/EDMA radičov.

3.3 Vývojový KIT DSK TMS320C6416

Vývojový KIT DSK TMS3206416 od firmy Texas Instruments (ďalej len DSK) je zariadenie určené na edukačné a vývojové účely. Pre širšie zavedenie do praxe nieje primárne určený a preto sa s ním v žiadnom komerčnom produkte nestretneme. Svoje uplatnenie avšak nachádza na samom začiatku procesu vývoja produktov na báze embedded systémov – vývoj, development. Po ukončení vývoja aplikácie na vývojovom KITe, sa výrobcovi zašle špecifikácia zariadenia, ktoré by malo byť osadené DSP procesorom a ďalšími potrebnými prostriedkami aké boli použité na vývojovom KITe. Takto vyrobené špecifické zariadenie bude určené na ďalšie komerčné, alebo iné využitie.

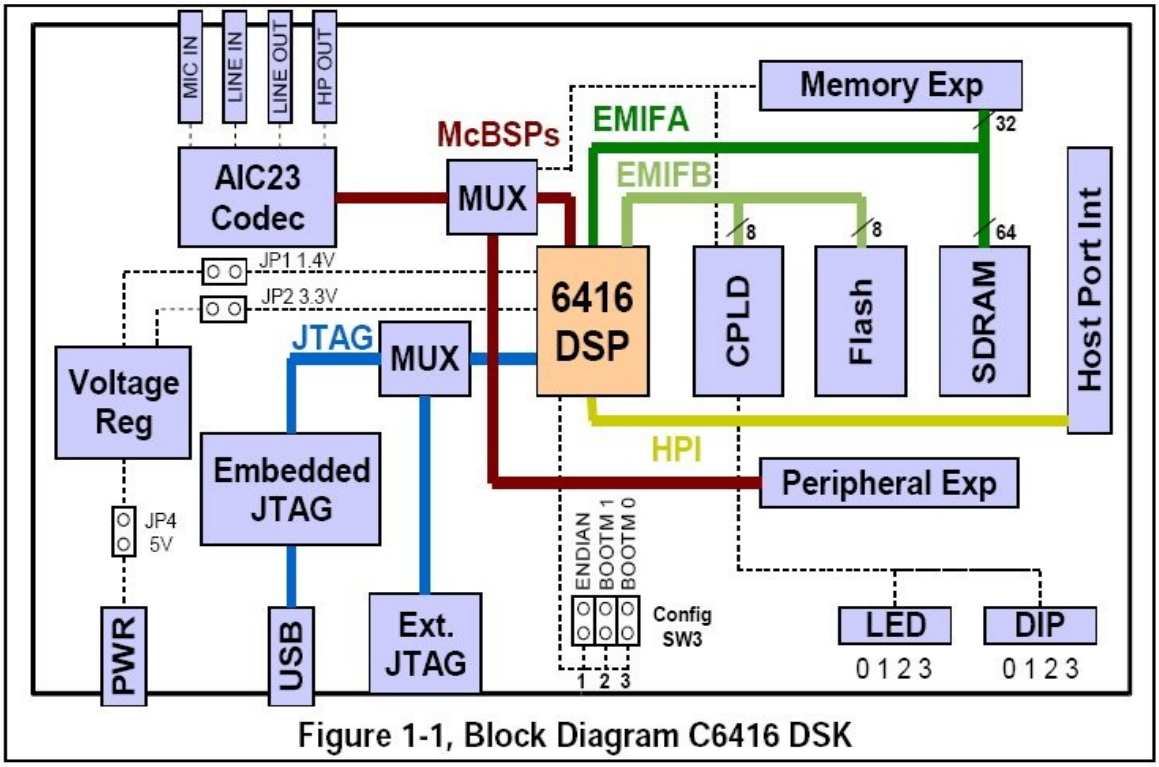


Figure 1-1, Block Diagram C6416 DSK

Hardware:

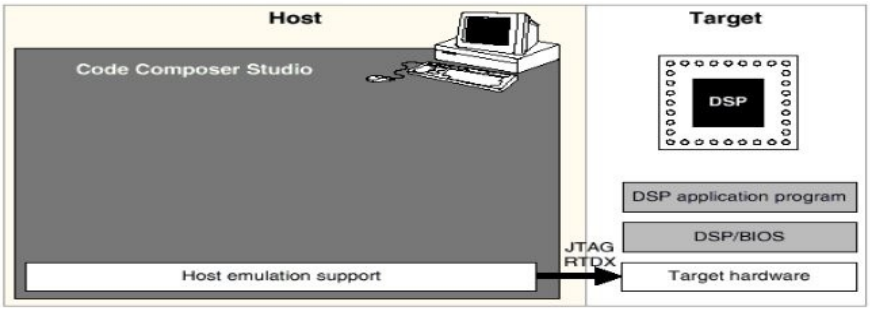
DSK obsahuje mnoho „on-board“ zariadení ktoré uspokoja potreby širokého spektra aplikácií, ktoré je možno na DSK vyvíjať.

- DSP procesor TMS320C6416 pracujúci na frekvencii 600MHz
- stereo audio codec AIC23
- 16MB DRAM
- 512KB non-volatile Flash pamäť
- 4 užívateľom ovládané LED a DIP prepínače
- softwarovo nastaviteľné konfigurácie karty cez registre
- nastaviteľné boot-ovanie
- slot na zapojenie prídavných kariet
- JTAG emulácia pre UBS port
- power supply +5V

DSP CPU a ostatné on-board periférie sú prepojené cez dve zbernice. 64-bitovú EMIFA a 8-bitovú EMIFB. AIC23 codec dovoľuje DSP prijímať a odosielať analógové signály. McBSP1 zbernica zabezpečuje nastavenie kodeku a McBSP2 slúži na prenos dát medzi kodekom a DSP. CPLD (programmable logic device) zabezpečuje zjednodušenie použitia jednotlivých komponentov na DSK. Napríklad použitím CPLD registrov, ktorými sa nastavujú jednotlivé zariadenia. 4 LED a 4 DIP prepínače poskytujú užívateľovi spätnú odozvu. CodeComposerStudio (vývojové prostredie, ďalej len CCS) komunikuje s DSK cez JTAG emulator s USB rozhraním.

Software:

DSK poskytuje okrem rôznorodého hardwarového vybavenia aj softwarové prostriedky, ktoré spolu vytvárajú veľmi silný nástroj pre vývoj aplikácií na DSP. DSK je totiž dodávaný s vlastným operačným systémom DSP/BIOS, ktorý zabezpečuje real-time scheduling, real-time analýzu (RTA), real-time data exchange (RTDX) a tak isto poskytuje jednoduché statické nastavenie systému (DSK) za pomoci grafického užívateľského rozhrania a CCS. Tak isto ako bežné operačné systémy aj DSP/BIOS poskytuje programátorovi služby, ktoré sú dostupné cez API.



3.4 Vývojové prostredie CCS

K softwarovému vybaveniu DSK patrí aj integrované vývojové prostredie nazvané Code Composer Studio – CCS, ktoré je spustiteľné len na platforme MS Windows. Toto vývojové prostredie poskytuje všetky potrebné nástroje, ktoré sú pri vývoji aplikácií nevyhnutné. To znamená, že vo všetkých fázach vývojového cyklu aplikácie si vystačíme s CCS.

V prvom rade sa jedná o nástroje správy projektov a editor zdrojových kódov, ktorý je plne integrovaný a spolupracuje s ostatnými nástrojmi. V spolupráci s prekladačom priamo zobrazuje inštrukcie assembleru postupne pre každý riadok zdrojového kódu, v spolupráci s debugovacími nástrojmi nastavuje breakpointy a probepointy, ….

Nástroje generovania kódu. Základ tvorí C/C++ kompilátor, ktorý prekladá štandardné ANSI C zdrojové kódy na inštrukcie assembleru optimalizované pre konkrétnu platformu DSP procesora použitého na DSK. Ďalej sem patrí assembler, ktorý preloží inštrukcie generované C/C++ kompilátorom do strojového kódu založeného na Common Object File Format (COFF). Na koniec prichádza Linker, ktorý vytvorí konečný spustiteľný súbor, ktorý sa nahrá do pamäte DSK.

Proces vývoja aplikácií môže byť výrazne uľahčený v prípade, ak má programátor k dispozícií kvalitné nástroje určené na debuging. CCS ich má hneď niekoľko. V prvom rade sú to BreakPointy, ktoré dovoľujú skúmať stav programu v jeho priebehu. O aktuálnom obsahu lokálnych a globálnych premenných, prípadne C/C++ výrazov nás informuje tzv. Watch Window. ProbePoint-y sa taktiež používajú k prerušeniu priebehu programu, avšak za účelom načítania vstupných dát zo súboru na PC do bufferu, zápisu výstupných dát do súboru na PC, prípadne k aktualizácii okien zobrazujúcich stav programu. Medzi ďalšie nástroje patrí vizuálna reprezentácia dát za pomoci Graph Window a iné.

Keďže vývojový KIT s DSP procesorom poskytuje len obmedzené prostriedky, veľmi dôležitým faktorom je efektivita vyvíjaných aplikácií. Optimalizačné procesy sú vo všeobecnosti veľmi zložitým problémom a v prípade aplikácií pre DSP to platí dvojnásobne. V CCS je k týmto účelom určený optimalizačný nástroj Profiler, ktorý programátorovi podáva dôležité informácie o tom, aký dlhý čas strávil procesor v kritickom mieste ktoré nás zaujíma.

Všetky nástroje, ktoré akýmkoľvek spôsobom potrebujú komunikovať priamo s DSK využívajú výhody real-timeového jadra CCS. To zabezpečuje real-time interaktivitu medzi CCS a DSP/BIOSom bežiaceho na strane DSK.

4. Implementácia rozpoznávača na DSP

4.1. Popis použitého rozpoznávača

V tejto časti sa pozrieme na samotné riešenie zadaného projektu. Začal by som tak trochu od konca a hneď na začiatok by som uviedol, ktorá fáza vývoja rozpoznávača reči bola cieľom tohto projektu. Po naštudovaní teórie rozpoznávania totiž vieme, že problematika rozpoznávania je komplikovaná záležitosť a vývoj kompletného a fungujúceho rozpoznávača na embedded systéme zaberie viac času ako rozsah jedného ročníkového projektu. Úlohou bolo adaptovať vybrané nástroje STK toolkitu na DSP. Prvú vec ktorú bolo treba implementovať je základ každého rozpoznávača – parametrizácia, alebo tzv. Feature Extraction. Pochopiteľne bez metód, ktoré sú schopné parametrizovať vstupný rečový signál, nemá význam sa ďalej zaoberať rozpoznávaním. Prvotným cieľom teda bolo adaptovať Feature Extraction. Keď už máme k dispozícií parametre, k rozpoznávaniu ďalej potrebujeme natrénované slová, resp. ich modely, ktoré chceme rozpoznávať. Trénovanie patrí v procese rozpoznávania k najzdĺhavejším procedúram. Kvalitné systémy rozpoznávania trénujú nad objemom dát, ktorý sa rádovo pohybuje v GB. Aj jednoduchšie rozpoznávače avšak pre vyhovujúcu rozpoznávaciu schopnosť potrebujú trénovať nad desiatkami až stovkami MB dát. Z tohto dôvodu a ďalej z dôvodu, že by to nemalo žiadny prínos, procedúra trénovania slov nebola na DSP portovaná a všetko trénovanie sa robí externe na PC. Na DSP sa prenesú už len natrénované modely. Poslednou časťou projektu bolo adaptovať rozpoznávač. V tejto fáze vývoja sa jedná len o jednoduché rozpoznávanie reči a ticha. To znamená, že potrebujeme natrénovať jeden model, ktorý bude reprezentovať ticho a druhý model, ktorý bude reprezentovať reč. Posledná poznámka k tejto časti sa týka spôsobu akým bol riešený vstup rečového signálu do systému. Výhodou hardwarovej platformy na ktorú bol systém portovaný je, že obsahuje audio codec schopný prijímať vstupné signály cez mikrofón. Avšak z dôvodu, že výsledkom projektu je akási testovacia fáza vývoja rozpoznávača reči, vstupné dáta sú privádzané do systému priamo cez USB rozhranie pripojeného k PC. Rozpoznávanie teda nieje real-timeové. Podrobnosti budú popísané v odstavci „Hardware engine“. Konečným výsledkom projektu je teda rozpoznávač na DSP schopný rozlíšiť ticho a reč.

Feature Extraction

V teoretickej časti tohto dokumentu sme sa zoznámili s viacerými možnými spôsobmi parametrizácie vstupného signálu. Na vstup sa privádza rečový signál, ktorý je vzorkovaný na frekvencii 8kHz. Táto vzorkovacia frekvencia je pre potreby rozpoznávania reči postačujúca. Vstupné dáta musíme ďalej rozdeliť do rámcov. Zvolená bola štandartná dĺžka, ktorá sa bežne používa v praxi a tou je 25ms. 25ms pri 8kHz vzorkovacej frekvencii je presne 200 vzoriek na jeden rámec. Posun medzi jednotlivými rámcami je 10ms (teda 80 vzoriek) z čoho plynie, že susedné rámce majú 160 spoločných vzoriek. Následne sa na každý rámec aplikuje Hammingovo okno. Z hotových rámcov môžeme počítať parametre. V tomto projekte som sa rozhodol pre parametre MFCC. Na začiatku si definujem banku filtrov: počet jednotlivých filtrov (23) a parametre každého z nich (výpočet parametrov možno nájsť v odkazovanej literatúre). Na hotový rámec aplikujem Fourierovu transformáciu, ktorá mi spočíta frekvenčné spektrum rámca definované komplexnými číslami. Vypočítam si modul týchto komplexných čísiel umocnený na druhú a na takto získané spektrum aplikujem banku filtrov. Dôvod aplikácie banky filtrov sme si vysvetlili v teoretickej časti. V tomto momente nám jeden celý rámec popisuje 23 neplnohodnotných parametrov. K získaniu MFCC potrebujeme ešte týchto 23 čísiel logaritmovať a aplikovať diskrétnu kosínusovú transformáciu, na ktorej výstupe získame 12 parametrov. Posledný trinásty MFCC koeficient získame ako normovanú strednú hodnotu dvanástich predošlých koeficientov.

Recognition

Rozpoznávač, ako už bolo spomenuté, má byť schopný rozpoznať či sa na vstupe objavila nejaká reč, alebo ticho. Na výber bolo opäť viacero možností typu rozpoznávania (viď teoretickú časť). Ja som si vybral rozpoznávač založený na štatistickom modelovaní, založený na Hidden Markov Models. Jednoduché rozpoznávanie typu – reč:ticho – je za pomoci HMM pekne realizovateľný. Modely reprezentujúce -reč:ticho- môžeme definovať ako jednostavové HMM, ktoré budú obsahovať len jedno Gaussovo rozloženie pravdepodobností. Rozpoznávanie sa teda obmedzí len na výpočet dvoch funkcií hustoty rozdelenia pravdepodobnosti (pre každý model jedna). Priamym porovnaním výsledkov identifikujeme, či sa na vstupe objavila reč, alebo ticho.

Trénovanie

Trénovanie modelov portované na DSP nebolo. Z tohto dôvodu všetku činnosť týkajúcu sa trénovania bolo za potreby naprogramovať na PC. Keďže programovacím jazykom použitým pri programovaní DSP je C, pochopiteľne aj nástroje pre trénovanie modelov na PC bolo treba implementovať v jazyku C. Rozdiely medzi kompilátorom na strane DSP (od Texas Instruments) a kompilátorom na PC (GCC) by nemali byť výrazné a v konečnom dôsledku by mali produkovať kód, ktorého výsledkom, jak na strane DSP, tak na strane PC, by mali byť rovnaké dáta. Avšak problém teoreticky môže nastať pri použití knižníc a ich funkcií dodávaných spolu s prekladačom na strane DSP. Konkrétne sa tento problém vyskytol pri feature extraction a počítaní rýchlej Fourierovej transformácie. Fourierova transformácia je operácia, ktorá výpočetne celkom náročná, avšak našťastie firma Texas Instruments poskytuje vlastné fix-pointové riešenie výpočtu

FFT. Dodávaná funkcia, ktorá počíta FFT je napísaná v assemblery a je do najväčšej možnej miery optimalizovaná pre konkrétnu hardwarovú platformu. Z tohto dôvodu dosahuje maximálnu efektivitu a rýchlosť výpočtu. Pochopiteľne, že túto konkrétnu funkciu nieje možné použiť na PC a preto pri programovaní nástrojov pre trénovanie je potrebné použiť funkcie výpočtu FFT programované klasicky pre PC. Pri následnom testovaní som prišiel k zisteniu, že výsledky dodané funkciami FFT na DSP a PC, sú do určitej miery odlišné. Rozdiely sa najviac prejavil pri následnom počítaní modulu umocneného na druhú. V ďalšom kroku testovania som sa avšak dopracoval k záveru, že v konečnom dôsledku pri porovnaní výsledných MFCC koeficientov, na strane DSP a PC, v predošlom kroku vzniknuté rozdiely do väčšej miery úplne zanikli. Týmto som teda dosiahol, že výsledky výpočtov získané na PC môžu byť bez problémov prenesené a použité na DSP.

Proces samotného trénovania je z dôvodu použitého rozpoznávača „reč-ticho“ vcelku jednoduchý. Trénovaný vstupný signál sa parametrizuje (13 MFCC). Následne sa z každého vektoru parametrov vezme i-ty koeficient a vypočíta sa stredná hodnota a smerodajná odchýlka nad týmito koeficientmi. Toto sa opakuje 13-krát pre všetky zložky MFCC. Takto získame 13-rozmerné Gaussovské rozloženie pravdepodobnosti, ktoré bude vložené do HMM.

Týmto sme si zhrnuli spôsoby a metódy akými funguje rozpoznávač reči, ktorý bol v tomto projekte implementovaný na DSP. Pozrime sa ešte na spôsob akým to celé reálne funguje na HW. Celý systém by sme si mohli rozdeliť na dve horizontálne vrstvy ležiace jedna nad druhou. V prvom rade je to spodná vrstva tzv. HW engine, ktorá obsahuje všetky funkcie a nastavenia HW prostriedkov. Tieto prostriedky budú poskytovať svoje služby, druhej nadradenej vrstve tzv. SW engine. Vrchná vrstva obsahuje samotné riešenie rozpoznávača reči. Komunikácia medzi týmito dvoma vrstvami je vcelku jednoduchá a v skutočnosti pozostáva len z jediného Interrupt-u smerujúceho z HW enginu do SW enginu. Samozrejme, že toto prerušenie sprostredkováva operačný systém DSP/BIOS. Schematicky zobrazenú architektúru tohto riešenia možno nájsť v prílohe.

4.2. Hardware engine

Úlohou spodnej aplikačnej vrstvy sú inicializačné procedúry a obsluha hardwarových prerušení generovaných zariadeniami na DSP KITe.

Hlavnou funkciou vrstvy ako aj celého programu je `main()`. V prvom rade zabezpečí prvotnú inicializáciu dátových štruktúr a konštánt, ktoré budú použité v nadradenej aplikačnej vrstve – SW engine. Konkrétne sa jedná o vyčistenie všetkých bufferov, vytvorenie Hammingovho okna `createHamming()`, výpočet koeficientov banky filtrov `melBanks_init()` a výpočet konštánt použitých pri DCT, `DCT_init()`. V ďalšom kroku prichádza na rad inicializácia hardware-u. Funkcia `AIC23_setParams()` zapíše požadované nastavenia do registrov audio kodeku, ktoré zabezpečia správne fungovanie. Dôležitými parametrami vzhľadom k rozpoznávaniu reči sú hlasitosť vstupného kanálu a najmä vzorkovacia frekvencia, ktorá je nastavená na 8kHz. Ako sme si už povedali v 3.kapitole, dátový transfer medzi audio kodekom a DSP sa odohráva na sériovom rozhraní McBSP. `initMcbsp()` zabezpečí správne nastavenie. Ak by tento prenos mal byť plne v réžii DSP procesoru, zabralo by to veľkú časť jeho prostriedkov. Procesor sa preto odbremeňuje od tejto „manuálnej“ práce a činnosť týkajúca sa transferu dát z audio kodeku, sa prenecháva DMA kontroléru. Kodek dodáva vstupné dáta konštantnou rýchlosťou 8kHz. Samozrejme, že od aplikácie požadujeme aby pracovala real-time a teda riešením pre zabezpečenie priebežného spracovania vstupného signálu je použitie dvoch vstupných bufferov, PING a PONG.

Pokým sa jeden buffer plní vstupnými dátami, druhý je spracovávaný (SW engine). Veľkosť bufferov je dimenzovaná na 200 vzoriek. Pri frekvencii 8kHz je teda jeden buffer naplnený za 25ms. Z tohto faktu vyplýva, že aplikácia zostane real-time, len ak spracovanie dát prebehne v tomto časovom horizonte. Funkcia `initEdma()` nastavuje kontrolér EDMA. Vytvorí dve DMA konfigurácie (pre každý buffer) a zlinkovaním sa zabezpečí striedavé napĺňanie bufferov. Vytvorený DMA kanál sa ešte zaregistruje ku generovaniu prerušení DMA kontrolérom. Poslednou procedúrou funkcie `main()` je zápis do príslušného miesta Interrupt Mask registru k aktivácií IRQ od EDMA.

Druhou časťou tejto aplikačnej vrstvy je obsluha prerušení, alebo InterruptServiceRoutines. V aplikácií rozpoznávača reči sa nachádza len jedno hardwarové prerušenie, generované DMA kontrolérom pri naplnení jedného z dvoch bufferov PING alebo PONG. Po tom ako DSP/BIOS prijme toto prerušenie, zavolá ISR edmaHwi(). Jedinou funkciou tejto rutiny je generovanie softwarového prerušenia processBufferSwi() a predanie parametru, ktorý identifikuje buffer pripravený k spracovaniu.

4.3. Software engine

Rozpoznávanie reči je plne implementované práve v tejto aplikačnej vrstve. Proces rozpoznávania odštartuje v momente prvého softwarového prerušenia ISR rutina `processBuffer()`. Táto identifikuje za pomoci parametra buffer, v ktorom sú uložené vstupné dáta a je pripravený k spracovaniu. Pred samotným Feature Extraction je potrebné zabezpečiť rámce. Vieme, že jednotlivé rámce musia byť prekrývané a preto výber rámcov priamo z PING&PONG bufferu nieje možný, pretože niektoré rámce pokrývajú dáta z oboch bufferov. Riešením je teda kruhový buffer dostatočnej veľkosti, z ktorého si pohodlne vyberieme požadovaný rámec. Funkcia `getFeatures()` v prvom rade uloží vstupné dáta do kruhového bufferu a v následne sa pokúsi vytvoriť čo najviac úplných, 25ms (resp. 200 vzoriek) dlhých rámcov pomocou funkcie `getFrame()`. Vytváranie rámcov je plne v réžii tejto funkcie. V prípade, že nedokáže vytvoriť nasledujúci rámec z dôvodu nedostatku dát v bufferi, počká si na ďalšie softwarové prerušenie kým sa buffer opäť nedoplní. Vlastnosťou funkcie getFrame() je aj to, že rámce automaticky násobí Hammingovým oknom. Po vytvorení jednotlivých rámcov prichádza najdôležitejšia funkcia programu.

Funkcia `processFrame()` je hlavným motorom aplikácie a preto si ju preberieme trochu podrobnejšie. Objasníme funkciu jednotlivých blokov, použité dátové typy a ich rozsahy a s pomocou Profiler-u CCS určíme aký výpočetný výkon DSP procesoru sa v jednotlivých blokoch spotrebúva.

V prvom rade si pre názornosť povieme, v akých intervaloch je táto funkcia volaná. A to najmä z toho dôvodu, že sa budeme zaoberať aj zaťažením procesora a bolo by vhodné mať predstavu koľko prevedených výpočtov sa ukrýva za percentami záťaže procesora. Každých 25ms je naplnený jeden buffer o veľkosti 200 vzoriek. Úplne na začiatku, z prvých 25ms vstupného signálu sme schopný vytvoriť len jeden rámec. Každý úplný rámec predstavuje zavolanie funkcie `processFrame()`. Po uplynutí 25ms získame ďalších 200 vzoriek, z ktorých postupných prekrývaním získame 2 úplné rámce a jeden polovičný 1. Ďalších 200 vzoriek a 3 úplné rámce. Týmto spôsobom je každých 25ms vytvorených striedavo 2 a 3 úplné rámce, tj. každých 25ms je zavolaná funkcia processFrame() dva alebo tri krát. Z tohto si môžeme utvoriť predstavu, koľko výpočtov je nutné previesť za časový interval 25ms. A teraz priamo k funkcii. Na vstup tejto funkcie sú dodávané 25ms rámce vstupného rečového signálu. Doteraz sme si však nepovedali, ako

vlastne tento vstupný signál vypadá. Audio codec plniaci funkciu A/D prevodníku je nakonfigurovaný na vzorkovaciu frekvenciu 8kHz a jednotlivé vzorky sú normované na rozsah `signed short`, teda +-32767. Sme vo východiskovej pozícií pred spracovaním rámca, keď vo vektore `frame` o veľkosti 200 vzoriek máme uložený rámec. Uveďme si teda ako sme na tom s vyťažením procesora. Doteraz sme vykonali kopírovanie bufferov, delenie na rámce a násobenie Hammingovým oknom. Tieto operácie a všetka réžia DSP/BIOS-u zamestnáva procesor na 1,06%. Nasleduje výpočet MFCC koeficientov.

Prvým krokom je FFT. Ako už bolo povedané, používa sa optimalizovaná funkcia DSP_fft32x32() napísaná v assemblery, ktorá sa vyznačuje vysokou efektivitou. Vstupom je vektor 256 komplexných čísiel `FFT_vector`, kde reálne zložky tvorené jednotlivými hodnotami rámca sa striedajú s nulovými imaginárnymi zložkami, na konci doplnené nulami. Po prevedení FFT máme výsledné spektrum uložené vo vektore `FFT_out`. Hodnoty typu Integer striedavo reprezentujú reálne a imaginárne zložky komplexných čísiel. Keď sme hovorili o efektívnosti použitého algoritmu FFT, dokázať nám to v tomto momente môže Profiler. Pri meraní vyťaženosti procesora sa hodnota ustálila na hodnote 1,57%.

Po získaní spektra potrebujeme vypočítať jednotlivé moduly komplexných čísiel umocnené na druhú za pomoci funkcie `fPower()`. Táto funkcia je bohužiaľ v tejto fáze vývoja projektu trochu problematická a nie celkom bezpečná. V teoretickej rovine ide o to, že dve dvojice 32 bitových čísiel majú byť násobené, medzi sebou sčítané a opäť uložené na 32 bitoch (Re*Re + Img*Img). Ak by sme chceli zabezpečiť úplnú bezpečnosť tejto funkcie, 32 bitové čísla by museli byť prevedené na 16 bitové, tzv. „scaling“ a vtedy by sme výsledok mohli bezpečne zapisovať do 32 bitov. Problém riešenia je avšak v tom, že spektrum čísiel, nad ktorými sa funkcia počíta, sa rozprestiera relatívne blízko okolo nuly, najväčšie čísla sú zriedkakedy definované na viac ako 16 bitoch. Požitím „scalingu“, konkrétne bitového posunu doprava o 16, by sme prišli o veľkú väčšinu dôležitých informácií, ktorých nositeľmi sú práve menšie čísla. Experimentovaním som dospel k záveru, že „scaling“ bitového posunu o 4 (tj funkcia div16) by mal v dostatočnej miere eliminovať možné problémy a informačný obsah z globálneho hľadiska by mal zostať nezmenený. Pri ďalšom vývoji projektu sa touto funkciou bude treba ešte zaoberať. V krátkom zhrnutí: funkcia `fPower()` prijíma vektor komplexných čísiel. Vezme reálnu a imaginárnu zložku každého čísla, prevedie „scaling“ (bitový posun doprava o 4), zložky umocní na druhú a sčíta. výsledok uloží na výstup. Pri pohľade na Profiler, vidíme mierny nárast zaťaženia procesora o približne 0,3%.

V ďalšom kroku aplikujeme na frekvenčné spektrum banku filtrov pomocou funkcie `applyMelBanks()`. Táto funkcia vytvára okrem iného aj akúsi pomysliteľnú hranicu v tom, akým spôsobom sa pozeráme na dáta. Doteraz sme totiž pracovali výhradne s celočíselnými dátovými typmi. Tento pohľad sa mení, pretože v tomto momente začíname používať dátové typy s pohyblivou rádovou čiarkou. Pripomeňme si, že použitie reálnych čísiel je zaujímavé z toho dôvodu, že hardwarová architektúra je navrhnutá ako fix-pointová. Nemá teda floating-point funkčné jednotky a práca s reálnymi číslami je iba emulovaná. Dôvod prechodu na pohyblivú rádovú čiarku je v jednoduchosti použitia týchto čísiel v nasledujúcich algoritmov. Celkové dôsledky tohto kroku budú zhrnuté v závere. Vráťme sa teda naspäť k funkcii `applyMelBanks()`. Na jej vstup priložíme prvú polovicu frekvenčného spektra (keďže druhá polovica je zrkadlovým obrazom) a aplikujeme banku 23 filtrov, ktorých koeficienty sme si predpočítali na začiatku vo funkcii `main()`. Vynásobením spektra a koeficientov filtrov a následnou sumou cez jednotlivé hodnoty získame 23 koeficientov vo vektore `Mel_feat`. Opäť sa pozrieme na Profiler. Nárast zaťaženia predstavuje 2% , čo znamená celkové zaťaženie na úrovni 3,92%.

`fLn()` je funkcia, ktorá vezme 23 koeficientov z predchádzajúcej kroku a prevedie logaritmus nad týmito číslami. Relatívne jednoduchá operácia dokáže na fix-pointovej architektúre

v spojení z reálnymi číslami situáciu celkom skomplikovať. Pri pohľade na výsledky Profiler-u veľmi rýchlo zistíme, že nárast zaťaženia o viac ako 3% na celkových 7,13% nieje zanedbateľný. Možnosti zlepšenia tohto nepriaznivého stavu preberieme v záverečnej kapitole.

Poslednými dvoma krokmi potrebnými k získaniu MFCC koeficientov sú funkcie `applyDCT()` a `fCalcC0()`. Výpočet DCT je podobne ako v prípade `applyMelBanks()` odľahčený použitím predpočítaných konštánt, ktoré už netreba zdĺhavo zakaždým počítať. Výsledkom DCT je prvých 12 MFCC koeficientov. Posledný 13 získame ako normovanú strednú hodnotu predchádzajúcich 12 koeficientov použitím funkcie `fCalcC0()`. Výsledok je uložený vo vektore `feats`. V tomto momente disponujeme plnohodnotnými MFCC koeficientmi. Hodnota zaťaženia procesora pri parametrizácii vstupného signálu sa ustálila na hodnote 9,01%.

Pre náš jednoduchý rozpoznávač, ktorý identifikuje ticho a reč, potrebujeme ešte okrem dvoch natrénovaných modelov aj funkciu LogGaussPDF(). Táto funkcia nám vyhodnotí pravdepodobnosť (presnejšie logaritmus pravdepodobnosti) vysielania vypočítaného vektoru MFCC koeficientov, daným stavom HMM, resp. Gaussovským rozložením. Parametrami funkcie sú okrem MFCC parametrov aj Gaussovské rozloženie pravdepodobnosti, reprezentované natrénovanými strednými hodnotami a smerodajnými odchýlkami. Výpočet zaťaží procesor ďalšími 5%. Pre rozpoznanie reči, alebo ticha budeme potrebovať s každým rámcom vyhodnotiť túto pravdepodobnosť pre oba modely.

5. Testovanie

Pri testovaní navrhnutého riešenia, bolo nesmierne dôležité overiť správnu funkčnosť každého jednotlivého bloku rozpoznávača reči. Z tohto dôvodu prebiehalo testovanie „offline“, a dáta boli dodávané na vstup priamo z PC cez USB rozhranie. Týmto spôsobom bolo možné pozorovať odozvy jednotlivých blokov systému na špecificky zadané vstupné dáta a tými istými dátami verifikovať jednotlivé algoritmy na strane PC napríklad za pomoci Matlabu alebo krátkych programov napísaných v jazyku C.

Pri testovaní systému ako celku bolo za potreby použiť konkrétne rečové signály. Trénovanie modelov prebiehalo na krátkom 20 sekundovom rečovom zázname na ktorom sa vystriedali dvaja rečníci, a rovnako dlhom zázname ticha. Po priložení zvukového signálu na ktorom sa nachádzalo striedavo reč a ticho, rozpoznávač pri týchto modelových podmienkach dokázal tieto dva segmenty identifikovať.

6. Záver

Krátke zhrnutie dosiahnutých výsledkov. Hlavným cieľom projektu bolo adaptovať vybrané nástroje použité pri rozpoznávaní reči na vhodnú DSP architektúru. V prvom rade bolo za potreby vyriešiť parametrizáciu rečového signálu. Na strane DSP som implementoval algoritmy pre počítanie MFCC parametrov. Tak isto bolo potrebné tento nástroj pre parametrizáciu naprogramovať aj na strane PC a to z toho dôvodu, že identické MFCC koeficienty sa musia použiť aj pri trénovaní modelov. Proces trénovania totiž zostal implementovaný len na strane PC. Natrénované modely sa manuálne prenesú do DSP, kde sú pripravené k použitiu. Rozpoznávač bol vyvinutý do takej podoby, že momentálne je schopný identifikovať vo vstupnom signále reč a ticho.

Systémy rozpoznávania reči sú výpočetne náročné aplikácie. Na začiatku procesu vývoja systému, pri výbere vhodnej hardwarovej architektúry nebolo možné presne určiť, do akej miery bude zvolená architektúra schopná obsluhovať a zvládať systém rozpoznávania reči. Ako si teda počínal vývojový KIT DSK TMS320C6416 ?

Už vo 4.kapitole sme si za pomoci Profiler-u uviedli hodnoty záťaže CPU v jednotlivých krokoch.

CPU load		
Overall	Partial	
1.00%		buffering, rámce, hamming. + réžia
processFrame()		
1.60%	0.60%	FFT preset + FFT()
1.90%	0.30%	fPower()
3.80%	1.90%	applyMelBanks()
7.10%	3.30%	fLn()
8.90%	1.80%	applyDCT()
9.00%	0.10%	fCalcC0()
14.20%	5.20%	LogGaussPDF()

Tabuľka nám poskytuje prehľad zaťaženia po jednotlivých etapách. Pripomeňme si, že funkcia processFrame() je volaná 2-3 krát každých 25ms. Pri tomto konkrétnom spôsobe riešenia nám DSP procesor poskytuje priestor napríklad na výpočet ďalších približne 17 funkcií hustoty rozdelenia pravdepodobnosti LogGaussPDF(). Keď vezmeme do úvahy ďalej napríklad Viterbiho algoritmus a iné metódy nevyhnutne potrebné k implementácií nie zložitého key word spotting-u, výpočetný výkon DSP procesora pri použití nezmenených metód nebude pravdepodobne dostatočný.

V ďalšom pokračovaní vývoja tohto systému bude bezpodmienečne nutné zmeniť niektoré prístupy k výpočtom jednotlivých častí. Pri pohľade do tabuľky je prvým kandidátom funkcia logaritmus. Riešením by mohlo byť použitie look-up tabuľky s predpočítanými hodnotami logaritmov. Podobné riešenie by malo znížiť obrovskú výpočetnú náročnosť aj v prípade funkcie LogGaussPDF(). Po optimalizácií týchto problematických procesov predpokladám, že neobsadená výpočetná kapacita procesora sa zvýši na takú úroveň, ktorá bude dostatočná pre ďalšie algoritmy a metódy a zabezpečí bezproblémové fungovanie výsledného key word spotting systému. Riešenie problému s nedostatočným výpočetným výkonom, ktorým by bolo odstránenie všetkých čísel s pohyblivou rádovou čiarkou a zmena algoritmov tak, aby pracovali s celými číslami, by bolo

komplexným riešením, ktorému sa pravdepodobne budeme môcť vyhnúť a nebudeme ho našťastie potrebovať. Ďalej bude potrebné ešte naprogramovať Viterbiho algoritmus, zmeniť štruktúru modelov rozpoznávaných slov (momentálne jednostavový HMM) a po offline otestovaní riešenia, zapojiť audio codec a vstupné signály privádzať do systému cez mikrofón. Týmto spôsobom by sme mali dopracovať k systému rozpoznávania reči, ktorý by bol portovaný na embedded system. Na ceste k dosiahnutiu tohto cieľa bude potrebné vyriešiť ešte množstvo problémov.

Literatúra a iné zdroje informácií:

[1] Lawrence Rabiner, Biin-Hwang Juang: Fundamentals of Speech Reocognition, Prentice Hall

[2] Doc.Dr.Ing Černocký Jan: materiály k prednáškam Číslicové spracovanie reči

(obrázky v teoretickej časti dokumentu boli prebraté práve z týchto materiálov)

[3] Schwarz P., Matějka P., Černocký J.: Towards Lower Rates in Phoneme Recognition, VUT, Brno

[4] Schwarz Petr: Feature Extraction algoritmy

Príloha:

Schematický popis riešenia programu: