VITERBIJEV ALGORITEM ZA DSP PROCESORJE

**Srečo Plevel, Tomaž Javornik, Igor Ozimek,
Roman Trobec and Gorazd Kandus
Institut Jožef Stefan, Ljubljana, Slovenia**

Ključne besede: sistemi komunikacijski digitalni, naprave radijske mobilne, kodiranje, dekodiranje, VITERBI algoritem, DSP obdelava signalov digitalna, DSP procesorji, kodiranje konvolucijsko, dekodiranje trdo, dekodiranje mehko, TMS320C4x Texas Instruments DSP procesorji, TMS320C6x Texas Instruments DSP procesorji

Povzetek: Sodobni digitalni komunikacijski sistemi, zlasti mobilne radijske naprave, so v vse širši uporabi. Za kvaliteten prenos podatkov in učinkovito izrabo razpoložljivega radijskega frekvenčnega spektra so potrebni kvalitetni modulacijski in kodirni postopki. Ena od pomembnih tehnik kodiranja je konvolucijsko kodiranje in dekodiranje z uporabo Viterbijevega algoritma. Modulacijski in kodirni postopki so relativno zahtevni za obdelavo v realnem času in so se običajno izvajali v specializiranih vezjih. Razvoj vedno bolj zmogljivih univerzalnih signalnih procesorjev pa omogoča programsko izvedbo teh postopkov in s tem koncept t.i. programljivega radia. Ta prinaša razne prednosti, med drugim enostavno spreminjanje in s tem prilagodljivost raznim komunikacijskim sistemom in standardom. V članku je predstavljena izvedba Viterbijevega algoritma za dve družini univerzalnih signalnih procesorjev TMS320C4x in TMS320C6x. Opisani so razni načini optimizacije in pohitritve Viterbijevega algoritma ter predstavljene dosežene zmogljivosti algoritma pri izvajanju na teh dveh procesorjih tako za trdo kot mehko dekodiranje.

Viterbi Algorithm for DSP Processors

Keywords: digital communication systems, mobile wireless devices, coding, decoding, VITERBI algorithm, DSP, Digital Signal Processing, DSP processors, convolution coding, hard decoding, soft decoding, TMS320C4x Texas Instruments DSP processors, TMS320C6x Texas Instruments DSP processors

Abstract: Modern digital communication systems, especially mobile wireless devices, are playing more and more important role in everyday life. For reliable data transmission and efficient utilisation of limited frequency spectrum resources, bandwidth-efficient modulation and coding must be employed. An important coding technique is convolutional coding and decoding with Viterbi algorithm. Modulation and coding procedures are relatively demanding for digital real-time processing and until recently they were mostly implemented in specialised integrated circuits. The development of ever more powerful general-purpose signal processors came to the point, where software implementation of these procedures is viable. This approach, known as software radio, has a number of advantages. It allows easy modification and adaptation to various communication systems and standards. In the article, the implementation of the Viterbi algorithm is described for two families of general-purpose digital signal processors, TMS320C4x and TMS320C6x. Various optimisations are presented together with the resulting performance in terms of achievable bit rates and error bit rates for hard and soft Viterbi decoding.

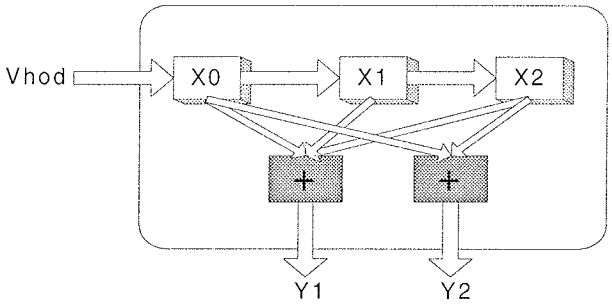
1. Uvod

V sodobnih digitalnih telekomunikacijskih sistemih se uporabljajo za zmanjšanje verjetnosti bitne napake kanalni kodirni postopki. Ti vnašajo v podatkovni tok dodatne bite, ki sprejemniku omogočajo popravljanje napak pri prenosu. Kanalne kodirnike delimo v bločne in konvolucijske. Pri bločnem kodiranju razdelimo bitni tok na posamezne bloke in vsakemu dodamo dodatne (redundantne) bite. Dekodiranje (s popravljanjem napak) se izvaja neodvisno na vsakem bloku posebej, nakar se izhodni biti sestavijo nazaj v neprekinjen bitni tok, seveda z zakasnitvijo, ki je posledica sprejemanja in obdelave posameznega bloka. Pri konvolucijskih kodirnih postopkih poteka kodiranje in dekodiranje sproti na neprekinjenem bitnem toku. Posamezen simbol na izhodu kodirnika je odvisen od trenutnega vhodnega simbola in od stanja kodirnika, to pa od določenega števila predhodnih vhodnih simbolov. Število izhodnih bitov je večje od vhodnih za nek faktor, kar zagotavlja redundanco in s tem možnost odpravljanja napak. Za dekodiranje konvolucijsko kodiranih signalov se uporabljajo različni algoritmi, najpogosteje pa Viterbijev algoritem. Na tržišču obstaja vrsta integriranih vezij za dekodiranje konvolucijsko kodiranih signalov, vendar je večina od njih vezana na določen konvolucijski kodirnik, torej na določen komunikacijski sistem. V zadnjih nekaj letih se je na področju mobilnih komunikacij pojavil koncept programljivega radija, za katerega je značilno, da se radijski vmesnik menja glede na storitve, ki jih zahteva uporabnik, in stanje radijskega kanala med sprejemnikom in oddajnikom. Eden glavnih gradnikov v sistemu programljivega radija so digitalni signalni procesorji - DSP. Viterbijev algoritem smo priredili tako, da ga je mogoče izvajati v digitalnih procesorjih družine Texas Instruments TMS320C4x in TMS320C6x.

2. Opis konvolucijskega kodiranja

Konvolucijsko kodiranje bomo prikazali na preprostem primeru kodirnika z enim vhodom in dvema izhodoma, pri katerem je izhodni 2-bitni simbol odvisen od trenutnega vhodnega bita in dveh predhodno poslanih bitov. To pomeni, da ima kodirnik dva registra, torej $2^2 = 4$ možna notranja stanja. Bločna shema kodirnika je prikazana na sliki 1. Konvolucijske kodirnike pogosto označujemo z zapisom (št. izhodnih bitov, št. vhodnih

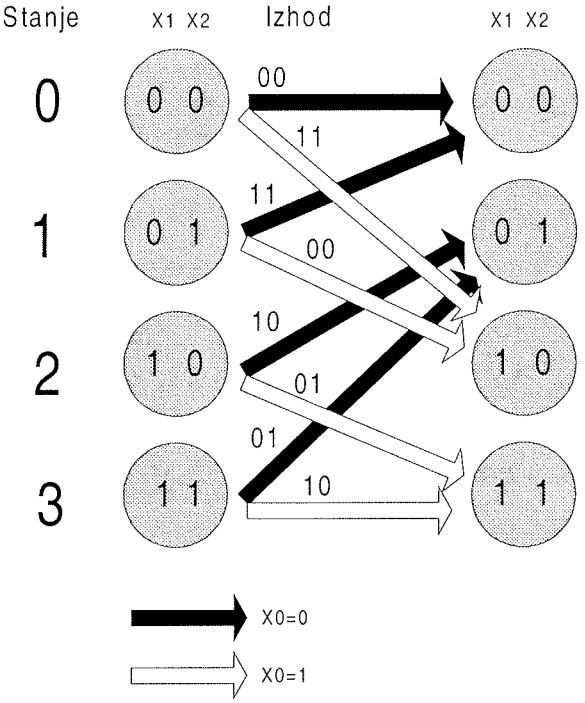
bitov, št. registrov), v našem primeru je oznaka (2,1,2). Ta zapis ne navaja povezav med zakasnilnimi členi in izhodi, zato z njo kodirnik še ni enolično določen.



Slika 1: Konvolucijski kodirnik (2,1,2)

Registra X1 in X2 predstavljata notranje stanje kodirnika, register X0 pa je vhod v kodirnik. Ko je na vhodu v kodirnik nov bit, se shrani v skrajno levi register (na sliki X0), vsi ostali pa se pomaknejo za eno mesto v desno. Indeks notranjega stanja predstavlja kar binarni zapis registrov X1 in X2.

Diagram prehajanja stanj prikazuje slika 2. Črne puščice pomenijo prehod stanja pri vhodnem bitu 0, bele pa pri vhodnem bitu 1. Binarni zapis v krogih pomeni stanja binarnih registrov pomnilnika konvolucijskega kodirnika. Iz diagrama prehajanja stanj je razvidno, da na vsakem koraku nov vhodni bit pride na najpomembnejše binarno mesto (z leve strani), najmanj pomemben bit (desni) pa odpade.



Slika 2: Diagram prehajanja stanj konvolucijskega kodirnika (2,1,2)

Če označimo bita notranjega stanja *x1* in *x2*, bit na vhodu pa *x0*, potem izhod kodirnika zapišemo z naslednjima enačbama:

$$y1 = x0 + x1 + x2$$
$$y2 = x0 + x2$$

Pri tem pomeni znak + vsoto po modulu 2 oziroma operacijo *izključujoči ali* (*xor*). Pri vsakem prehodu iz trenutnega v naslednje stanje sta izhodna bita kodirnika enolično določena s prehodom, njuna vrednost je v sliki 2 zapisana nad vsakim prehodom. Izhodni bitni tok opisanega konvolucijskega kodirnika ima dvojno število bitov v primerjavi z originalnim nekodiranim podatkovnim bitnim tokom na vhodu in zahteva zato dvakrat večjo hitrost prenosa v zameno za večjo zanesljivost.

3. Viterbijev dekodirnik

Za dekodiranje konvolucijsko kodiranega signala se običajno uporablja Viterbijev algoritem. (Ta se uporablja tudi v primerih, ko podatki sicer niso konvolucijsko kodirani, pač pa neidealna prenosna karakteristika komunikacijskega kanala povzroča enak učinek, t.j. medsebojni vpliv med podatkovnimi biti oz. simboli.) Na prenosnih poteh prihaja do napak. Če so napake na prenosni poti statistično neodvisne in se ne pojavljajo v rafalih (burstih), je postopek dekodiranja konvolucijsko kodiranega signala s pomočjo Viterbijevega algoritma optimalen. Rafalom napak, ki pogosto nastopajo v telekomunikacijskih sistemih, se izognemo s premešanjem (scrambling) poslanih bitov v daljšem časovnem intervalu.

Dekodirnik opravi svojo nalogo, če ugotovi, kako so prehajala notranja stanja v kodirniku, ko je kodiral podatke za prenos. Po dogovoru se kodirnik na začetku nahaja v stanju 0 (vsi biti notranjega pomnilnika so nič).

Viterbijev algoritem gradi usmerjen graf vseh možnih poti v mrežnem diagramu iz začetnega stanja 0. Mrežni diagram dobimo, če prehajanje po diagramu stanj raztegnemo po časovni osi. Vsako vozlišče predstavlja notranje stanje ob določenem času, vsaka povezava pa predstavlja možen prehod med notranjimi stanji kodirnika. Vsaki povezavi izračunamo Hammingovo razdaljo med sprejetim simbolom in izhodnim simbolom kodirnika pri izbrani povezavi. (Hammingova razdalja je preprosta aritmetična vsota napak, v binarnem primeru je to kar število napačnih bitov.) Minimalna pot v grafu predstavlja najbolj verjetno prehajanje stanj kodirnika. Dolžina vsake poti pomeni natanko število napačnih bitov.

Viterbijev algoritem uporablja princip dinamičnega programiranja, ki pravi, da si je potrebno za vsako vozlišče v grafu zapomniti le najboljšo pot do njega.

Druge poti v vozlišče se zavrže. Viterbijev algoritem za vsako vozlišče izračuna kumulativno metriko vseh poti, ki vodijo v vozlišče. V vsakem vozlišču ohrani le najboljšo pot (pot z najmanjšo metriko). Na koncu izmed vseh vozlišč poišče vozlišče z minimalno metriko in iz opisa poti v to vozlišče določi izhodni simbol dekodirnika.

Čeprav ima konvolucijski kodirnik končno dolžino (število registrov), bi moral dekodirnik, ki opravlja recipročno funkcijo, za teoretično optimalno dekodiranje v vsakem trenutku računati metriko celotne poti od začetka oddajanja signala. To v praksi seveda ni mogoče, saj mora imeti dekodirnik končno dolžino. Napaka glede na optimalno rešitev postane zanemarljiva, če je dolžina dekodirnika (število upoštevanih vozlišč na poti pri vsakokratnem izračunu) enaka ali večja od približno štirikratne vrednosti števila registrov kodirnika.

4. Uporaba tabel za hitrejše delovanje Viterbijevega algoritma

Delovanje Viterbijevega algoritma pohitrimo z uporabo tabel. Število in vrsta tabel je odvisna od velikosti pomnilnika, ki je na razpolago, in velikosti konvolucijskega kodirnika.

Za začetek smo se odločili, da implementiramo Viterbijev algoritem v C-ju na enem procesorju. V članku /1/ je predstavljen pristop s pomočjo vnaprej izračunanih tabel, ki jih algoritem uporablja za hitrejše računanje. Optimizacija je usmerjena v hitrost izvajanja, velikost porabljenega pomnilnika pa ni pomembna. Velikost tabel eksponentno narašča s številom registrov kodirnika, ki pa je relativno majhno (tipično manjše od 10).

Uporabljene so sledeče tabele:

Tabela vseh možnih prehodov med stanji *MOrder* (Mesh Order):
Prva pomembna tabela je tabela vseh možnih prehodov med stanji za vsak vhodni simbol. Vrednost tabele predstavlja stanja, iz katerih lahko pridemo v dano stanje. S pomočjo te tabele v trenutku dobimo vsa stanja, iz katerih je možen prehod v neko stanje, kar je ključnega pomena pri hitrem iskanju najbolj verjetne poti.

Tabela vnaprej izračunanih vseh možnih Hammingovih razdalj - *Distan*:
To je najpomembnejša tabela, v kateri so shranjene vse možne Hammingove razdalje za vsako pot pri vsakem vhodnem simbolu. Tabela je tri-dimenzionalna. Prva dimenzija pove, za kateri vhodni simbol gre, druga pove, za katero pot po vrsti gre, tretja pa, za katero končno stanje. Branje iz te tabele nadomesti relativno počasno sprotno računanje razdalj, kar pomeni večkratno pohitritev izvajanja dekodirnega algoritma.

Tabeli trenutnih minimalnih razdalj do vozlišč - *OldMetric* in *NewMetric*:
Potrebujemo še dve tabeli, ki vsebujeta trenutno metriko (minimalno Hammingovo razdaljo) do vsakega notranjega stanja.

Tabela optimalnih prehodov med vozlišči - *NewSurviv*:
V tabeli je shranjenih zadnjih 32 poti, ki so preživele za vsako vozlišče. Vsakič pomaknemo bite v tabeli za eno mesto v levo, če preživi druga pot, besedo zapolnimo z desne z 1, sicer pa z 0. Tako se po 32 korakih v njej nahaja 32 bitov, ki po vrsti hranijo informacijo, katera pot je preživela.

Delovanje dekodirnika, ki ga ponazarja diagram na sliki 3, je sledeče:

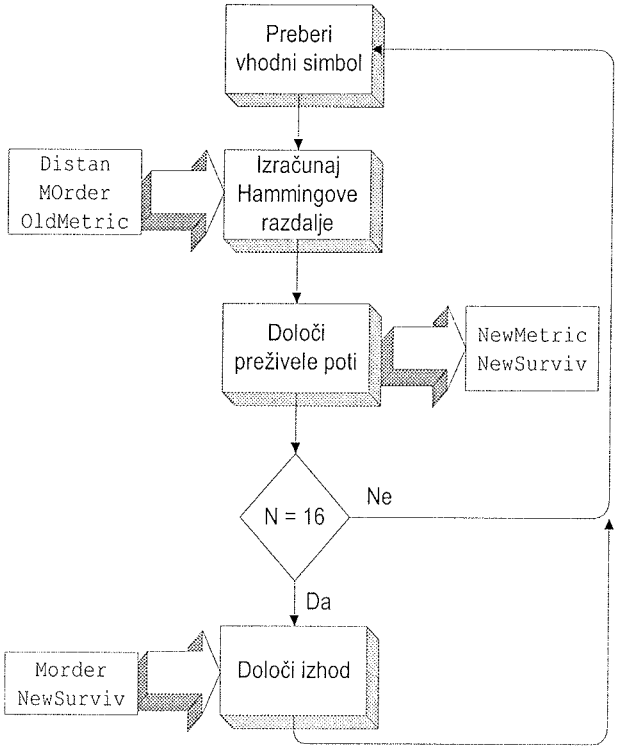
Preberemo vhodni simbol, imenujmo ga *Symb*.

Hammingove razdalje računamo preko vseh notranjih stanj. Na vsakem koraku prideta v vsako vozlišče dve poti, od katerih ena preživi, druga pa odpade. Če računamo stanje *state*, potem nam *MOrder[state][0]* ter *MOrder[state][1]* povesta obe stanji, iz katerih vodi pot v diagramu prehajanja stanj v končno stanje *state*. Imenujmo ju *s0* in *s1*. Stari metriki stanja *OldMetric[s0]* je potrebno prišteti še Hammingovo razdaljo za zadnjo pot *Distan[Symb][0][state]*. Isto ponovimo za drugo pot (*OldMetric[s1]*, pri kateri prištejemo *Distan[Symb][1][state]*).

Preživelo pot in s tem novo metriko do vozlišča *state* dobimo tako, da manjšo vsoto zapišemo v *NewMetric[state]*. Bite v *NewSurviv[state]* vsakič premaknemo za eno mesto v levo, najbolj levi bit odpade, z desne pa se zapiše nova vrednost, ki pove, katera pot je preživela na zadnjem koraku.

Določanje izhoda se izvede na vsakih 16 sprejetih simbolov. Algoritem najprej poišče stanje z minimalno metriko, nato pa s pomočjo tabele *NewSurviv* izračuna najbolj verjetno poslane podatke. Izpisuje se le 16 podatkov, ki so bili poslani pred 32 simbolnimi intervali. To pomeni, da vedno izpisujemo dekodirano zgodovino za 16 do 32 bitov nazaj, kar ustreza 32-bitni besedi procesorja. Celotna potrebna zgodovina preživelih poti za vsako stanje je shranjena v eni besedi tabele *Newsurviv*. Dekodirnik se mora najprej premakniti 16 prehodov stanj v zgodovino brez izpisovanja izhodnega simbola, nato pa še 16 z izpisovanjem.

Koda je napisana splošno za poljubne konvolucijske kodirnike z enim vhodom in dvema izhodoma. Pri povečevanju števila registrov kodirnika se eksponentno veča število notranjih stanj, s tem pa velikosti tabel in čas izvajanja.



Slika 3: Viterbijev dekodirnik

3.2. DSP algoritem prirejen za TMS320C44

Hitrost Viterbijevega algoritma smo testirali na procesorju Texas Instruments TMS320C44. Kot testni primer smo izbrali konvolucijski kodirnik z enim vhodom, dvema izhodoma in petimi zakasnilnimi celicami (2,1,5), ki se uporablja za kanalno kodiranje pri prenosu govora ali podatkov v GSM telekomunikacijskih sistemih. Izhodna hitrost vokoderja uporabljenega v GSM mobilnih telefonih je 13 kbit/s, (blok 260 bitov, dolžina bloka je 20 ms). Blok 260 bitov je razdeljen na 3 dele, od katerih sta konvolucijsko kodirana le prva dva podbloka dolžine 182 bitov. Prvemu podbloku dodamo tri paritetne bite. Zaradi lastnosti konvolucijskega kodirnika je potrebno celotnemu bloku dodati še štiri ničle na koncu prvih dveh podblokov, tako da je podatkovna hitrost na vhodu konvolucijskega kodirnika 9,4 kbit/s za en govorni kanal. Pri prenosu podatkov s prenosno hitrostjo 9,6 kbit/s uporabljamo enak konvolucijski kodirnik, vendar je v tem primeru vhodna podatkovna hitrost v konvolucijski kodirnik 12,2 kb/s.
Pri taktu DSP procesorja 50 MHz in algoritmu napisanem v programskem jeziku C je procesor potreboval za izpis 16 bitov informacije 0,305 ms, kar pomeni podatkovno hitrost 52,4 kbit/s.

3.2.1 Uporaba hitrega notranjega pomnilnika

V naslednjem koraku smo vse tabele prestavili v hitri notranji pomnilnik, ki ga imajo procesorji TMS320C44 na naslovih od 0x002FF800 do 0x002FFFFF. Velikost naslovnega prostora je 8 kB, oziroma 2048 lokacij. Branje in pisanje v notranjem pomnilniku traja samo en strojni cikel, v običajnem pomnilniku pa dva strojna cikla. V tem primeru dobimo hitrost 72,9 kbit/s.

3.2.2 Uporaba naslavljanja »post incremented«

Dostop do tabel preko kazalcev predstavlja naslednjo izboljšavo. Uporabili smo posebno naslavljanje (*post incremented addressing*), pri katerem se pri linearnem prehodu skozi tabelo ne izgubi nič časa za naslavljanje tabele, saj se register (ki predstavlja kazalec na tabelo) ob branju vrednosti, kamor kaže, samodejno poveča za ena. RISC procesorji za opisani ukaz porabijo le en cikel. Vsaka od vrstic v spodnjem primeru sešteje staro metriko stanja in metriko dane poti, hkrati v istem ukazu pa še poveča oba kazalca za 1, tako da zaradi posebne razvrstitve elementov tabel v naslednjem koraku že dostopa do naslednjega stanja in poti:

```
ADDI   *AR2++,*AR7++,R10
ADDI   *AR2++,*AR7++,R9
```

V dveh urinih periodah procesorja dobimo dolžini obeh možnih poti v končno stanje. Podatkovna hitrost se je povečala na 78,3 kbit/s.

3.2.3 Implicitna uporaba vrednosti *MOrder*

Dodatno povečanje hitrosti dosežemo z implicitno uporabo vrednosti v tabeli *MOrder*. Za izbrani konvolucijski kodirnik opazimo, da se vrednosti v tabeli *MOrder* za prvo polovico stanj in prvo ter drugo vhodno vejo v stanje povečujejo linearno s korakom 1. Enako velja za drugi del stanj konvolucijskega kodirnika.

Tab. 1: Tabela MOrder

Pot\stanje	0	1	2	3	4	5	6	7
0	0	2	4	6	8	10	12	14
1	1	3	5	7	9	11	13	15

Pot\stanje	8	9	10	11	12	13	14	15
0	0	2	4	6	8	10	12	14
1	1	3	5	7	9	11	13	15

Lastnost tabele smo izkoristili tako, da vrednosti kazalca nismo brali iz tabele *MOrder*, temveč smo ga linearno povečali. Z opisano izboljšavo smo dosegli za 10% hitrejše delovanje algoritma in s tem podatkovno hitrost 86,7 kbit/s.

3.2.4 Uporaba razporeditve dimenzij v tabeli *Distan*

V tabeli *Distan* je v prvi dimenziji vhodni simbol, tako da so v pomnilniku vse vrednosti za isti vhodni simbol skupaj (32 vrednosti = 16 stanj × 2 poti). Algoritem v enem koraku vse razdalje po poteh računa za isti

vhodni simbol, ki ga sprejme kot parameter. Na začetku algoritem nastavi odmik kazalca na pravo vrednost, nato se samodejno ob vsakem naslavljanju poveča za 1. Zaradi omenjenega načina naslavljanja ne porabimo dodatnega časa. Dosegli smo propustnost 93 kbit/s.

3.3 Mehki Viterbijev dekodirnik

Opisani dekodirniki so trdi dekodirniki, ker obdelujejo le vrednosti 0 in 1. V vseh komunikacijskih sistemih, tudi digitalnih, se podatki v komunikacijskih kanalih prenašajo s pomočjo signalov, ki so analogne veličine. S pretvorbo signala v logične vrednosti (0,1) izgubimo del informacije, ki bi lahko izboljšal postopek dekodiranja. Če ničlo in enico predstavimo z vrednostjo, ki pove, kako blizu je sprejeti podatek logični enici ali ničli, in te vrednosti obdelujemo v dekodirniku, govorimo o mehkem Viterbijevem dekodirniku. Pri 8-bitni A/D pretvorbi sprejme dekodirnik za vsak prenesen simbol vrednost med 0 in 255. Vrednost 0 pomeni čisto ničlo, vrednost 255 pa čisto enico. Dekodirnik (2, 1, *n*) mora sprejeti vrednosti za dva sprejeta bita (t.j. skupno 16 bitov) za vsak dekodirani bit. Računanje razdalj je bolj zamudno, saj bi bilo za enak način uporabe tabel kot pri prejšnjem dekodirniku potrebno imeti zelo velike tabele. Tabela *Distan* bi bila velika kar $16\times2\times2^{16}$ = 2M naslovov, saj je sedaj 2^{16} možnih vhodnih simbolov. Če pa bi npr. uporabili le vrednosti od 0 do 15 (4-bitna A/D pretvorba), potem bi bila velikost 16×2×256 = 8K naslovov.

Pri mehkem Viterbijevem dekodirniku se Hammingova razdalja nadomesti s kvadratom evklidske razdalje (razlike sprejetega simbola in pravilnega simbola) za dano pot. Minimalna vsota po poti pomeni najbolj verjetno oddano zaporedje simbolov, torej je to Viterbijevo dekodiranje na podlagi maksimalne verjetnosti (*MLVD - Maximum likelihood Viterbi Decoding*, /7/, poglavje 5-1-4).

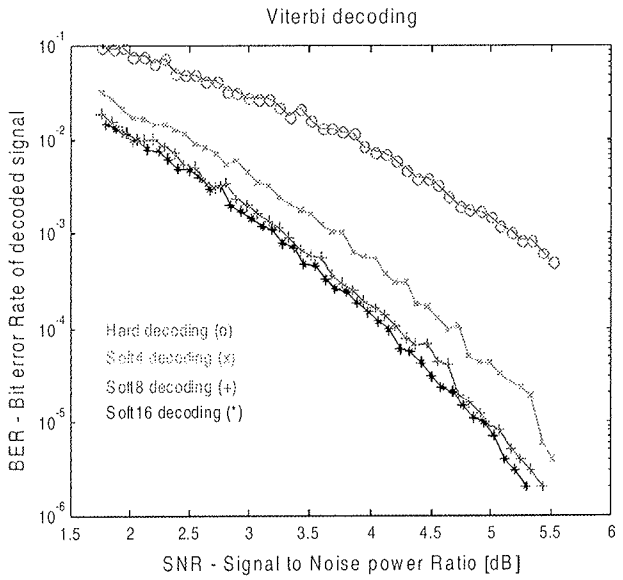
3.3.1 Hitro dekodiranje s 16 nivoji

Napisali smo program za hitro dekodiranje za 16 nivojski dekodirnik (N=16) z uporabo dveh novih tabel *Dist1[SIMBOL][STANJE][POT]* in *Dist2[SIMBOL][STANJE][POT]*. Tabeli imata enako vlogo kot jo je imela tabela *Distan* pri trdem dekodiranju, le da se sedaj prva tabela nanaša na prvi, druga pa na drugi sprejeti simbol. Vsaka ima dimenzijo vhodnega simbola enako 16, tako da sta velikosti 16×16×2 = 512 lokacij. Skupaj zasedeta 1024 lokacij, kar je že polovico hitrega pomnilnika. Ostale tabele so v delu druge polovice hitrega pomnilnika, nekaj pomnilnika pa mora ostati prostega, saj se tam po privzetem nahaja sklad. Funkcija za računanje razdalje je sedaj rahlo spremenjena, saj je potrebno prišteti stari metriki razdalji za oba sprejeta simbola.

Skupaj z vsemi opisanimi časovnimi optimizacijami in uporabo opisanih tabel preko kazalcev in hitrega pomnilnika smo dosegli hitrost mehkega dekodirnika 73 kbit/s, kar je več kot dvakratna pohitritev zaradi uporabe tabel in le 20% slabše od trdega dekodirnika.

3.4 Primerjava odpornosti proti šumu med mehkim in trdim dekodiranjem

Z manjšanjem šuma se razmerje med učinkovitostjo mehkega in trdega dekodiranja močno povečuje. Mehko dekodiranje je veliko učinkovitejše kot trdo dekodiranje.



Slika 4: BER diagram za različne Viterbijeve dekodirnike

Slika 4 prikazuje odvisnost pogostosti napak (BER – *Bit Error Rate*) od razmerja moči signala in šuma (SNR). Štiri krivulje prikazujejo: trdi dekodirnik, mehki dekodirnik s štirimi nivoji, mehki dekodirnik z osmimi nivoji ter mehki dekodirnik s šestnajstimi nivoji. Na sliki se lepo vidi, da je mehki dekodirnik veliko boljši od trdega. Z večanjem razmerja signal/šum pa pride razlika še bolj do izraza. Vsa merjenja so zaradi omejenega časa približna.

Mehki dekodirnik je za več razredov boljši od trdega dekodirnika, razmerje se s kvaliteto prenosnega kanala še povečuje. Vendar pa večanje natančnosti na več kot 16 nivojev (en prenesen bit predstavljen s šestnajst vrednostmi) ne prinese vidne izboljšave h kvaliteti dekodiranja. Tako med rezultati dekodirnika s 16 nivoji in dekodirnika z 256 nivoji pri Gaussovem šumu nismo zasledili razlike. Več nivojev pa pomeni mnogo večje tabele, kar lahko posledično na večini sistemov pomeni tudi počasnejše izvajanje zaradi omejene velikosti najhitrejšega pomnilnika (npr. notranji pomnilnik pri DSP, predpomnilnik pri PC).

Pri mehkem dekodiranju dobimo malo boljše rezultate, če simbolov 0 in 1 pri A/D pretvorbi ne postavimo čisto na meje amplitudnega območja pretvornika. Tako pri 16 nivojskem mehkem dekodirniku dobimo najboljše rezultate, če vrednost 0 predstavimo z vrednostjo 2, vrednost 1 pa z vrednostjo 13. Na ta način ostane šum znotraj amplitudnega območja pretvornika in ohrani svoje lastnosti, t.j. Gaussovo porazdelitev. V nasprotnem primeru bi bil šum na eni ali drugi strani porezan, ne bi bil več Gaussov, in zato tudi dekodiranje ne bi bilo več optimalno.

4. Zaključek

Hitrost trdega dekodiranja na 50 MHz procesorju zadošča za bazno postajo GSM sistema, ki mora delovati z 8-kratno hitrostjo mobilnih GSM terminalov. Za mehko dekodiranje pa je potrebno imeti približno 20% hitrejši procesor. Pri opisanem algoritmu se ne uporablja operacij s plavajočo vejico. Zaradi tega je mogoče uporabiti hitrejše in cenejše celoštevilčne DSP procesorje za izvedbo Viterbijevega algoritma v realnem času. Opisani algoritmi omogočajo realizacijo programljivega radijskega vmesnika, ki se prilagaja zahtevam uporabnika in pogojem prenosa radijskega signala med sprejemnikom in oddajnikom.

5. Literatura

/1/ S. M. Said, K. R. Dimond, »Realtime implementation of the Viterbi decoding algorithem on a high-performance microprocessor«, *Microprocessors and microsystems*, vol. 10, no. 1, january/february 1986

/2/ N. Pavešič, *Informacija in kodi*, Fakulteta za elektrotehniko, Ljubljana, 1997

/3/ Domen Šuligoj, Roman Trobec, Borut Robič, »Vzporedna izvedba Viterbijevega algoritma«, v tisku - *Elektrotehniški vestnik*, Ljubljana

/4/ G. Feygin, P. G. Gulak, P. Chow, »A Multi processor Architecture for Viterbi Decoders with Linear Speedup«, *IEEE Transactions on Signal Processing*, Vol. 41, No. 9, September 1993

/5/ I. N. Bronstein, K. A. Semendjajew, G. Musiol, H. Mühlig, *Matematični priročnik*, Tehniška založba Slovenije, 2. Predelana izdaja, Ljubljana, 1997

/6/ Texas Instruments, *TMS320C4x User's guide*, Digital Signal Processing Products, 1993

/7/ John G. Proakis, *Digital Communications*, McGraw-Hill International Editions, Electrical Engineering Series, 1995

Srečo Plevel
študent 4. letnika FRI
Institut Jožef Stefan, Jamova 39, Ljubljana
tel.: (1) 477-3900, fax.: (1) 251-9385, (1) 426-2102
email: sreco.plevel@campus.fri.uni-lj.si

Dr. Tomaž Javornik
Institut Jožef Stefan, Jamova 39, Ljubljana
tel.: (1) 477-3900, fax.: (1) 251-9385, (1) 426-2102
email: tomaz.javornik@ijs.si

Dr. Igor Ozimek
Institut Jožef Stefan, Jamova 39, Ljubljana
tel.: (1) 477-3900, fax.: (1) 251-9385, (1) 426-2102
email: igor.ozimek@ijs.si

Dr. Roman Trobec
Institut Jožef Stefan, Jamova 39, Ljubljana
tel.: (1) 477-3900, fax.: (1) 251-9385, (1) 426-2102
email: roman.trobec@ijs.si

Prof. Gorazd Kandus
Institut Jožef Stefan, Jamova 39, Ljubljana
tel.: (1) 477-3900, fax.: (1) 251-9385, (1) 426-2102
email: gorazd.kandus@ijs.si

Prispelo (Arrived): 20.10.00 *Sprejeto (Accepted): 22.11.00*