Uporaba Mikroprocesorjev

Študijsko gradivo

Fakulteta za matematiko in fiziko, Oddelek za fiziko
Dušan Ponikvar
Avgust 2010

Kazalo

1. Uvod

Fizik pogosto želi meriti in nadzirati fizikalne procese. Pri tem mora merilne postopke in načine poseganja v procese ter s tem povezano računanje poenostaviti in avtomatizirati do te mere, da intervencije operaterja niso potrebne. Osebni računalnik, bolj natančno procesor, ki je v osebni računalnik vgrajen, ponuja možnosti za računanje in avtomatizacijo, napisati je treba le primeren program. Če procesor opremimo s senzorji, ki fizikalne veličine pretvorijo v električne signale, lahko procesor naučimo »videti« fizikalni proces. Če ga opremimo z aktuatorji, ki električne signale spremenijo v fizikalne veličine, lahko procesor »spremeni potek« fizikalnega procesa. Za priključevanje tako senzorjev kot aktuatorjev na procesor so potrebni prilagodilni členi, ki jih imenujemo vmesniki (s tujko »interface«).

Za rešitev enostavnih nalog s področja avtomatizacije marsikdaj zadošča že enostaven procesorski sistem s procesorjem z nekaj malega elektronskih komponent, ki je prilagojen reševanju danega problema in je po svoji zgradbi, velikosti in porabi električne energije bistveno enostavnejši in cenejši od osebnega računalnika. Take procesorje najdemo v večini vsakdanjih elektronskih naprav, katerih delovanje upravljamo z vsaj nekaj gumbi, saj procesorji omogočajo bolj zanesljivo delovanje naprav in predvsem več možnosti za upravljanje ter interakcijo z uporabnikom.

Univerzalne in enostavne procesorske sisteme z mikroprocesorjem in vmesniki brez ohišja ali napajalnika se dobi na tržišču že za nekaj desetin € in imajo vse potrebne vhodne in izhodne priključke za senzorje in aktuatorje, zanje je potrebno napisati le še program, po katerem naj napravica deluje. Če imamo znanje in nekaj veselja lahko procesorske sisteme sestavimo tudi sami iz elektronskih komponent, ki so na tržišču na razpolago za še manj denarja.

Bolj kompleksne naloge rešujemo s pomočjo kompleksnejših procesorjev. Osebni računalnik, čeprav ni posebej prilagojen za zajemanje podatkov, je popularen predvsem zaradi relativno hitrega računanja, sposobnosti interakcije z uporabnikom ter splošne dostopnosti in relativno nizke cene. Zato ga dopolnimo z vmesnikom za zajemanje podatkov in napišemo program za obdelavo podatkov na osebnem računalniku. Kadar sposobnosti osebnega računalnika ne zadoščajo, ga opremimo z enoto, ki je hkrati vmesnik in še procesor za obdelavo. Na tam področju so zelo popularni digitalni signalni procesorji in programabilna vezja FPGA (»field

programmable gate array«), katerim prav tako definiramo postopke za reševanje problema ali avtomatizacijo v obliki programa, osebni računalnik pa služi kot ohišje, napajalnik in vmesnik za uporabnika.

Tudi kompleksnejša vezja lahko sestavimo iz osnovnih gradnikov in jih prilagodimo reševanju zastavljene naloge in se tako ognemo rabi osebnega računalnika. Ta pot je primerna takrat, ko potrebujemo večje število enakih enot, sicer cena vloženega časa in znanja ne opravičuje razvoja kompleksnega vezja.

V pričujočem zapisu želimo bralca poučiti z možnostmi, ki jih ponujajo sodobni procesorji in računalniki, ki so opremljeni z vmesniki za priključevanje senzorjev in aktuatorjev.

Začeli bomo z opisom števil, kot jih predstavljajo signali v procesorju ter definirali enote v sistemu za zajemanje podatkov. Enotam bomo določili lastnosti in merila za ovrednotenje kakovosti, kar bo posebej koristno pri odločanju o uporabi komercialnih modulov. Nadalje se bomo poučili o postopkih za priključevanje vmesnikov na procesor in senzorjev na vmesnik. Definirali bomo signale na vodilih, ki nastopajo v procesorskih sistemih, tako paralelnih kot serijskih.

Nadaljevali bomo z opisom centralne procesne enote mikroprocesorja in načinom pretoka podatkov med njenimi sestavnimi deli ter postopki za optimizacijo delovanja. Pojem centralne procesne enote bomo razširili na mikroprocesor in mikrokontroler ter opisali njihove sestavne dele s stališča strojne opreme.

Po strojni opremi bo na vrsti programiranje. Začeli bomo z naborom ukazov za izbrani mikroprocesor, hierarhijo jezikov in prevajalniki ter razporeditvijo prevedenega programa v spominu računalnika.

Osrednji del po posvečen zajemanju podatkov z dano strojno opremo. Zajemanje podatkov je lahko enkraten dogodek, lahko pa zajemanje periodično ponavljamo. Pri zadnjem je treba poskrbeti za nemoteno zajemanje kljub siceršnji zasedenosti procesorja z drugimi nalogami, zato uporabljamo prekinitvene tehnike, časovne intervale med zaporednimi zajemki pa definira števnik in signal ure. Pri tem bomo uporabili nekaj trikov, ki močno olajšajo shranjevanje zajemkov v polja ter temeljijo na binarnem zapisu števil.

Zajete signale je potrebno obdelati ali posredovati uporabniku, zato se bomo v nadaljevanju posvetili tema nalogama. Obdelali bomo digitalno filtriranje signalov po metodi FIR (»Finite Impulse Response«) in IIR (»Infinite Impulse Response«), za zadnje bomo potrebovali postopek, imenovan z-transformacija ter na tak način definirali diskretno prenosno funkcijo digitalnega filtra.

Zadnje čase se uveljavljajo novi elektronski sklopi, ki so namenjeni računanju in so bistveno hitrejši od procesorjev. To so digitalni signalni procesorji, ki so še vedno zgrajeni kor procesorji, vendar so optimirani za izvajanje operacij, potrebnih za digitalno obdelavo signalov. Še hitrejša so vezja integrirana vezja FPGA, ki so skupki logičnih vrat. Ta vrata načrtovalec - programer združuje tako, da opravljajo

zastavljeno nalogo. Zaradi velike količine logičnih vrat je možnih veliko vzporednih operacij, zato so FPGA vezja lahko hitra. Programiranje je še najbližje sestavljanju logičnih vezij iz logičnih vrat, čeprav se uporablja abstrakten programski jezik.

2. Zapisi števil v digitalnih sistemih

V digitalnem računalniku ima signal ali bit, ki ga prenašamo po posamezni žici, lahko vrednost nič (»LO«, 0, »FALSE«) ali ena (»HI«, 1, »TRUE«). Ker želimo računati z števili, ki imajo večji nabor vrednosti, več bitov združimo. Pri tem se držimo ideje decimalnega zapisa števil. Ko deset cifer od 0 do 9 ne zadošča, nadaljujemo z dvomestnimi števili tako, da levo zapisani cifri pripišemo desetkratno težo njene desne sosede ter tako zapišemo vrednosti od 0 do 99. Ko še to ne zadošča, dodamo levo tretjo cifro, ki ji spet pripišemo desetkratno teže njene desne sosede in tako naprej.

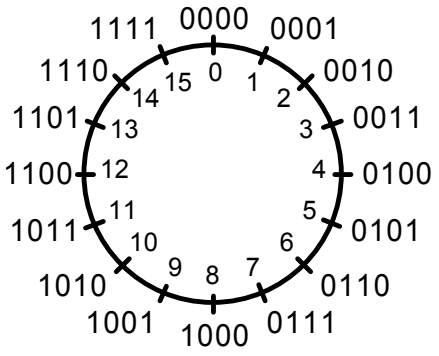
V digitalnem računalniku navadno združimo osem bitov (D_7, D_6, … D_0) v bajt. Pri tem ima levi sosed dvakratno težo desnega soseda. Kodiranje je podano na sledeč način.

$$D_7D_6D_5D_4D_3D_2D_1D_0 =$$
$$= D_7 \cdot 2^7 + D_6 \cdot 2^6 + D_5 \cdot 2^5 + D_4 \cdot 2^4 + D_3 \cdot 2^3 + D_2 \cdot 2^2 + D_1 \cdot 2^1 + D_0 \cdot 2^0$$

Zgled: $10110010_2 =$

$$= 1 \cdot 128 + 0 \cdot 64 + 1 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 178_{10}$$

Nabor števil binarnega zapisa prikažemo s številskim krogom, saj je pri omejenem številu bitov v zapisu cikličen. Vzemimo, da je število zapisano s štirimi biti, kot je to narisano na sliki 2.1. Zunanja števila predstavljajo binarno vrednost, notranja pa decimalni ekvivalent. Ko vrednost števila postopoma povečujemo, se ta spreminja 0000_2, 0001_2, 0010_2, 0011_2, 0100_2, 0101_2, 0110_2, 0111_2, 1000_2, 1001_2, 1010_2, 1011_2, 1100_2, 1101_2, 1110_2, 1111_2, 0000_2, 0001_2, … Zaradi te lastnosti je treba pri računanju s celimi števili vseskozi paziti, da rezultat računa ne prekorači obsega števil, saj bi ga lahko napačno interpretirali.



Slika 2.1: Številski krog za štiri bitni zapis

Sodo število bajtov nadalje

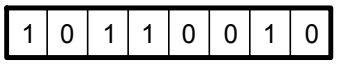
združujemo v besedo (»word«, dva bajta, šestnajst bitov) in dolgo besedo (»long word«, štirje bajti, 32 bitov). Najmanj pomemben bit na desni strani zapisa imenujemo LSB (»least significant bit«), najpomembnejši bit na levi strani zapisa pa MSB (»most significant bit«). Tabela 2.1 podaja nabor števil pri zapisih s tipično dolžino.

Število bitov	Število kombinacij	Obseg za pozitivna cela števila	Obseg za vsa cela števila
8	256	0 do 255	-128 do +127
16	65535	0 do 65535	-32768 do +32767
32	4294967296	0 do 4294967296	-2147483648 do +2147483647

Tabela #: Obseg števil pri različnih dolžinah zapisa

Ker je binarni zapis števila za človeka težje berljiv in razumljiv, uporabljamo še heksadecimalni zapis, ki ga označimo s črko $_H$. Po štiri bite binarnega zapisa združimo v eno cifro šestnajstiškega zapisa. Pri tem začnemo združevati z desne, najprej štiri najmanj pomembne bite. Cifre šestnajstiškega zapisa so ob 0_H do 9_H in od A_H do F_H, pri tem F_H predstavlja 15_{10}.

Zgled: $10110010_2 = B2_H$



Slika 2.2: Simbol za register za osem bitov z vpisano vrednostjo $B2_H$

Zapisu v obliki bajta, besede ali dvojne besede so v računalniku prilagojena vodila in registri ter aritmetične in logične enote. Register je tako sestavljen iz osmih, šestnajstih ali dvaintridesetih flip-flopov, slika 2.2.

Na zgoraj zapisan način lahko predstavimo samo pozitivna cela števila. Kadar potrebujemo še negativna cela števila, uporabimo zapis z dvojiškim komplementom. Tu najbolj pomemben bit predstavlja predznak, ostali biti pa vrednost. Kadar je MSB nič, je število pozitivno in preostanek zapisa predstavlja njegovo vrednost. Kadar je MSB ena, je število negativno in absolutno vrednost števila dobimo tako, da v preostanku zapisa vse bite komplementiramo (ničle spremenimo v enke in obratno) ter prištejemo ena.

Zgled: $34_{10} = 00100010_2 = 22_H$

Zgled: $-34_{10} = 11011110_2 = CF_H$

Procesor oziroma njegov uporabnik mora vedeti, v kakšnem zapisu so podana števila, sicer lahko rezultate matematičnih operacij napačno interpretira.

Nekateri pretvorniki analognih signalov v digitalne in obratno uporabljajo za negativna števila drugačno kodo, kjer najbolj pomemben bit zapisa predstavlja predznak, ostali biti pa absolutno vrednost števila. Tak zapis je manj primeren za računanje, ker vsebuje dve različni kombinaciji bitov za vrednost nič.

Poleg števil so v procesorju lahko kodirane tudi črke, cifre in ločila. Pri tem na nivoju procesorja navadno uporabljamo sedem bitno ASCII kodo (»American Standard Code for Information Interchange«), tabela je v prilogi, v osebnih računalnikih pa popolnejšo verzijo z imenom UNICODE.

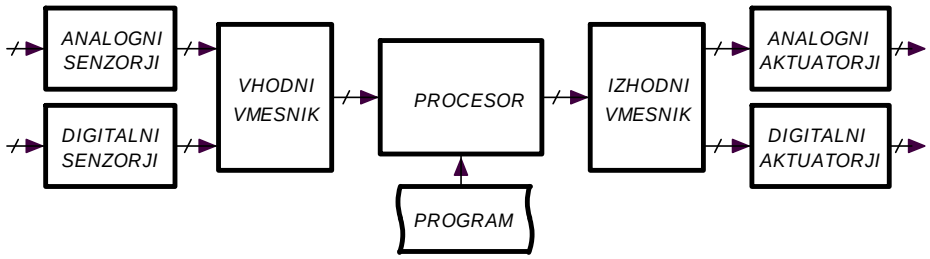
Celoštevične vrednosti zadoščajo za prenašanje podatkov med moduli procesorskega sistema, za računanje v procesorju pa se zaradi večje preciznosti uporabljajo števila s plavajočo vejico. Za komunikacijo s pretvorniki se vedno uporabljajo celoštevilčne vrednosti, zato zapisa s plavajočo vejico tu ne bomo podrobneje obravnavali. V višjih programskih jezikih so na razpolago ukazi za pretvorbo med zapisi.

Števila so v procesorju zapisana z nizi ničel in enic, te so predstavljene z napetostmi. Velikosti napetosti za logično nič in ena so odvisne od tehnologije izdelave procesorja in logičnih vezij ter napajalne napetosti vezij. Te izbere načrtovalec strojne opreme, ki vezje sestavi. Tipične vrednosti pri napajanju vezij s 5 volti so: logična nič je zastopana z napetostjo med nič do približno 1V, logična ena pa z napetostjo od vsaj dva volta do pet voltov. Vmesne napetosti niso dovoljene, kar zagotavlja visoko stopnjo varnosti pred napačnim interpretiranjem napetosti v logični nivo. Pri napajanju s 3,3 volta je logična nič predstavljena z napetostjo od nič do največ pol volta, logična ena pa z napetostjo, ki znaša med 1,5V in 3,3V. Podrobnejše specifikacije najde bralec v tovarniških podatkih za procesorje in logična vezja.

Spominske enote so v mikroprocesorskih sistemih še vedno večinoma sestavljene tako, da omogočajo shranjevanje posameznih bajtov. Ko shranjujemo na primer besedo, ki jo sestavljata dva bajta, na zaporedni mesti v spominski enoti shranimo posamezen bajt besede. Kadar na prvo mesto spravimo manj pomemben bajt, na naslednje pa bolj pomembnega, govorimo o zapisu »Little Endian«. Kadar spravimo bajta v nasprotnem vrstnem redu, govorimo o »Big Endian« zapisu. Zapisa sta enako dobra, seveda pa mora pisec programske opreme vedeti, za katero vrsto zapisa gre. Osebni računalniki, ki temeljijo na procesorjih firme Intel, uporabljajo večinoma »Little Endian«, Motorolini pa »Big Endian« zapis.

3. Sistem za avtomatizacijo s procesorjem

S procesorjem želimo avtomatizirati fizikalni proces, zato moramo procesorju omogočiti meritev pomembnih fizikalnih veličin, na podlagi teh s procesorjem izračunati potrebne korekcije za proces ter preko aktuatorjev v proces poseči. Poleg procesorja potrebujemo torej senzorje, vmesnike za senzorje, vmesnike za aktuatorje in aktuatorje. Tipična bločna shema verige za obdelavo signala je na sliki 3.1.



Slika 3.1: Digitalni sistem za avtomatizacijo fizikalnega procesa

Senzor spremeni fizikalno veličino v analogni električni signal. Pri tem je relacija med fizikalno veličino in senzorjem definirana s prenosno funkcijo senzorja. Ta je lahko statična, če je odziv senzorja trenuten, in dinamična, če se odziv senzorja s časom prilagaja vrednosti merjene fizikalne veličine. Podrobnejše lastnosti in principe delovanja senzorjev bralec najde pri predmetu »Fizikalna merjenja«, v pričujočem zapisu nas zanimajo le električne lastnosti senzorja.

Izhodna napetost senzorja je lahko neprimerno velika, zašumljena ali kako drugače neprimerna za obdelavo s procesorjem, zato jo je treba analogno prilagoditi v enoti za analogno obdelavo, ki je del vhodnega vmesnika. Predelani, še vedno analogni signal prevzame analogno-digitalni pretvornik, ki je prav tako del vhodnega vmesnika. Ta prevede signal v digitalno vrednost, ki jo lahko obdelujemo s procesorjem. Nekatere procese lahko nadzorujemo tudi brez pretvorbe, če senzorji že dajejo digitalne signale.

V procesorju teče program, ki na podlagi izmerjenih veličin računa potrebno korekcijo. Program napiše usposobljeni uporabnik - programer, ki obvlada postopek

reševanja problema in programiranje procesorja ter pozna strojno opremo, s katero ima opravka.

Izračunano korekcijo procesor posreduje v fizikalni sistem tako, da jo s pomočjo digitalno analognega pretvornika najprej pretvori v analogno napetost. To napetost vezje za analogno obdelavo po potrebi prilagodi po velikosti ali drugih lastnostih ter jo posreduje aktuatorju. Obe enoti sta del izhodnega vmesnika.

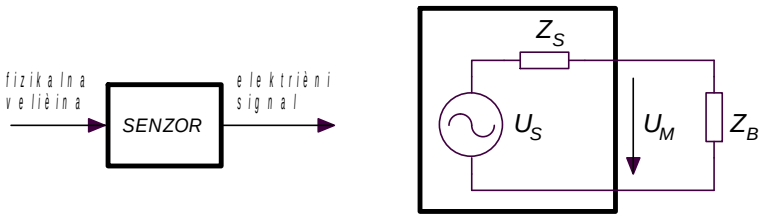
Aktuator dejansko poseže v fizikalni proces. Tipični aktuatorji so na primer ventili, grelniki, motorji… Tudi aktuatorju lahko pripišemo statične in dinamične lastnosti, vendar nas bodo tu zanimale le njihove električne lastnosti.

Kadar je sistem za avtomatizacijo enostaven, lahko katero od naštetih enot tudi izpustimo.

3.1. Senzorji

3.1.1. Električne lastnosti

Senzor predstavimo kot blok, slika 3.2 levo, njegova nadomestna električna shema je na isti sliki desno. Vse električne lastnosti senzorja nadomestimo z generatorjem U_S, ki daje napetost v odvisnosti od merjene fizikalne veličine po prenosni funkciji senzorja T_S(s) in notranjo impedanco senzorja Z_S. Kadar senzor meri počasi se spreminjajoče veličine, reaktivna komponenta notranje impedance senzorja ne vpliva na signal in takrat lahko izhodno impedanco reduciramo na izhodno upornost R_S.



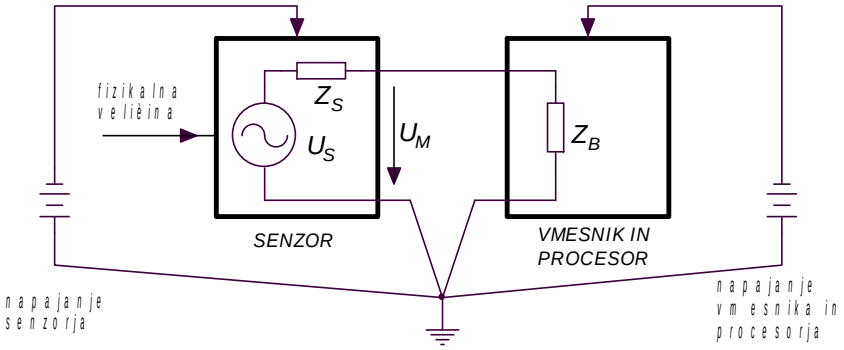
Slika 3.2: Senzor in njegova električna nadomestna shema z bremenom

Senzor priključimo v vezje oziroma na vhodni vmesnik, to vpliva na napetost U_M, ki jo namerimo med priključnima sponkama senzorja in jo vidi priključeno vezje. Vhodni vmesnik s stališča obravnave električnih signalov nadomestimo z bremenom Z_B, zato je napetost obremenjenega senzorja lahko drugačna od U_S. Breme vpliva tudi na prehodni pojav pri spremembi vrednosti fizikalne veličine. Večina industrijskih senzorjev je izdelana tako, da je izhodna impedanca majhna, breme pa le malo vpliva na signal s senzorja.

3.1.2. Napajanje in ozemljitve

Senzor pogosto potrebuje vir električne napetosti. Pri tem napajanje za senzor ni nujno isto kot za ostale module v merilnem sistemu. Ker senzor tiplje majhne spremembe fizikalne veličine, naj bo njegova napajalna napetost čim bolj enakomerna in brez motenj, ki bi lahko vplivale na merilni rezultat. Zato je napajanje

senzorja običajno ločeno od napajanja digitalnega vezja s procesorjem. Tipična povezava napajalnih napetosti je na sliki 3.3. Skupen ostaja le priključek ozemljitve, ki je skupna referenčna točka za izhodno napetost senzorja.

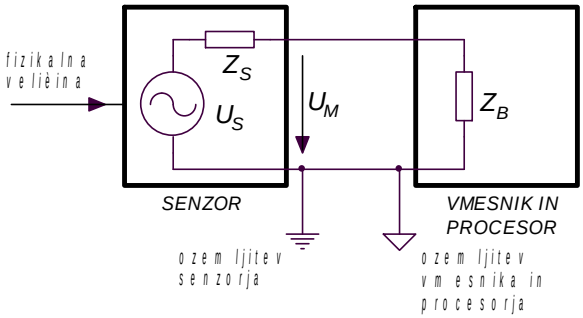


Slika 3.3: Zaradi ločenih napajanj senzorja in procesorja so motnje manjše

3.1.3. Načini priključevanja senzorjev

Senzor daje na svojem izhodu napetost U_M, ki je sorazmerna merjeni fizikalni veličini in je razlika potencialov med izhodno sponko senzorja in ozemljitvijo senzorja, slika 3.4. To napetost moramo privesti na vhod naslednje stopnje v merilnem sistemu, torej moramo za povezovanje uporabiti dve žici: za ozemljiitev in za signal.

Težave se pojavijo takrat, kadar so enote med sabo oddaljene. Po predpisih in predvsem zaradi varnosti morajo biti vse enote ozemljene tam, kjer so postavljene in zato ni nujno, da je ozemljitveni priključek pri senzorju na istem potencialu, kot ozemljitveni priključek pri procesorju. Če bi taka ozemljitvena priključka povezali z žico, bi po njej lahko tekli tokovi, ki bi motili ali celo uničili enoti.



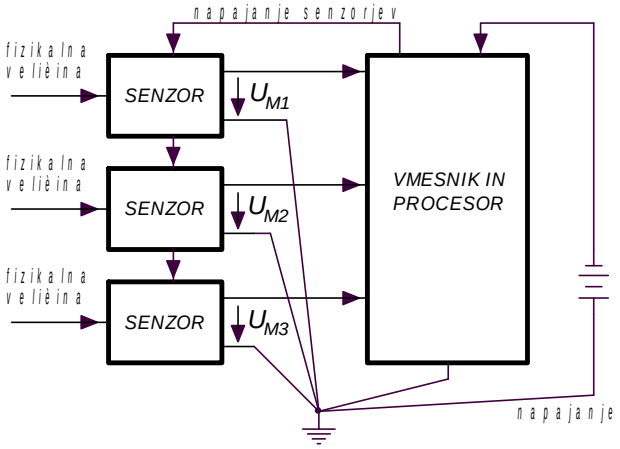
Slika 3.4: Težave z različnimi ozemljitvami

NRSE – »non referenced single ended«

Pri tem načinu priključevanja predpostavimo, da so zemlje vseh v sistem povezanih enot na natanko istem nivoju. To je res, ko po ozemljilnih žicah vseh enot,

ki so speljane v isto točko, tečejo le majhni tokovi, ki na teh žicah ustvarjajo zanemarljivo majhne padce napetosti. Pogoju je mogoče zadostiti le takrat, ko so senzorji blizu procesorju, sicer mora biti vsaka enota posebej ozemljena tam, kjer je postavljena. Na sliki 3.5 je bločna shema take vezave za tri senzorje, na vmesniku so narisani vhodni priključki zanje.

Če so senzorji in procesorski sistem blizu ter je napajanje procesorskega sistema mogoče filtrirati tako dobro, da ne povzroča motenj, lahko tudi senzorje napajamo s procesorskega sistema.



Slika 3.5: NRSE povezava senzorjev na vmesnik

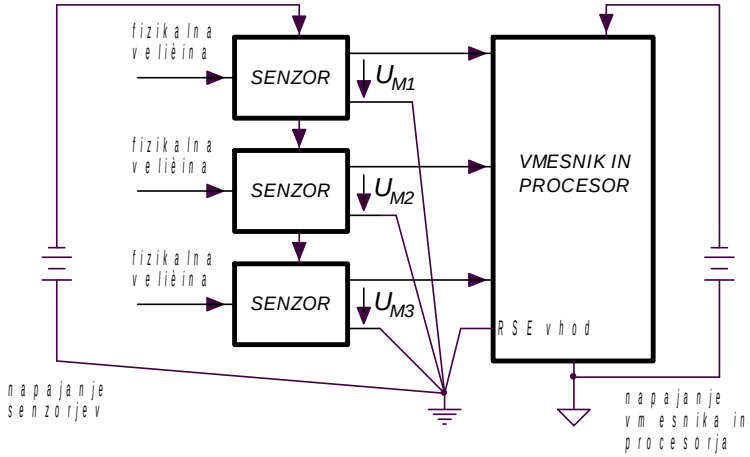
RSE – »referenced single ended«

Pri tem načinu povezovanja predpostavimo, da so ozemljitve vseh senzorjev, ki jih priključujemo v merilni sistem, na istem potencialu. Drugačna je le ozemljitev procesorskega sistema, zato moramo ozemljitveni potencial senzorjev po žici dovesti procesorskemu sistemu. Bločna shema povezav za tri senzorje je na sliki 3.6. Priključek za ozemljitev senzorjev *RSE vhod* na procesorskem sistemu ni povezan z ozemljitvijo procesorskega sistema, ampak služi le za referenco, od katere naj se meri vhodna napetost.

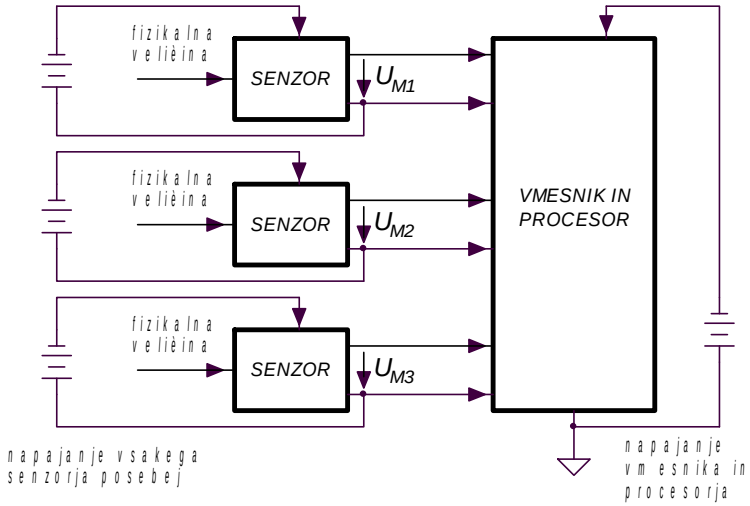
DIFF – »differential«

Kadar je senzorjev več in njihove ozemljitve niso na istem potencialu, moramo signal z vsakega senzorja posebej pomeriti kot razliko med izhodno napetostjo senzorja in njegovo ozemljitvijo, slika 3.7. Procesorski sistem z vmesnikom mora meriti razliko dveh potencialov, zato ima za posamezen senzor pripravljen par priključkov. Alternativno lahko s procesorskim sistemom posebej merimo potencial ozemljitve posameznega senzorja in potem še izhodno napetost vsakega senzorja ter izračunamo razlike v programu.

Nekateri pretvorniki analognih signalov v digitalne dopuščajo konfiguracijo svojih vhodov tako, da direktno pomerijo razliko dveh priključenih potencialov. Pri drugih potrebujemo analogna elektronska vezja, ki to razliko izračunajo in posredujejo naslednji enoti. V ta namen uporabljamo diferencialne in instrumentacijske ojačevalnike, glej poglavje 3.2.4.



Slika 3.6: RSE povezava senzorjev na vmesnik in procesor



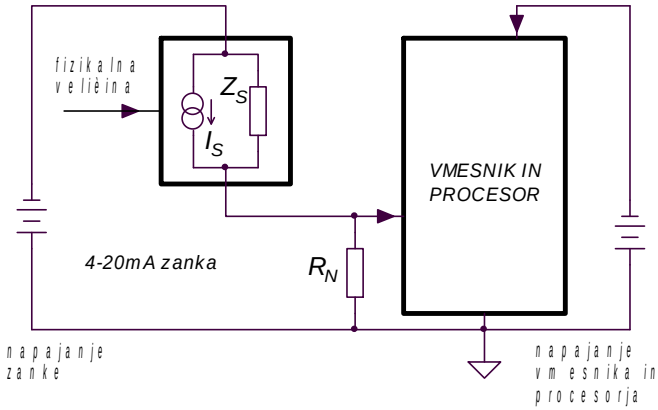
Slika 3.7: DIFF povezava senzorjev na vmesnik in procesor

3.1.4. Tokovna zanka – 4-20mA

Nekateri industrijski senzorji so opremljeni z vezjem, ki merjeno veličino pretvori v tok, zato je izhodna veličina takega senzorja tok in ne napetost. Govorimo o tokovni

zanki 4mA do 20mA; pri minimalni merjeni fizikalni veličini teče skozi senzor tok 4mA, pri maksimalni pa 20mA. Minimalni tok 4mA zadošča za napajanje elektronike, ki je vgrajena v senzor. Tipična povezava senzorja v merilni sistem je na sliki 3.8. Tok skozi senzor I_S povzroči na uporniku R_N padec napetosti, ki ga merimo z vmesnikom in procesorjem.

Za priključevanje senzorja s tokovnim izhodom potrebujemo le dve žici, kar močno poenostavi ožičenje. Tokovna zanka je tudi bistveno manj občutljiva na motnje iz okolice, saj v zanki inducirane napetosti ne morejo spremeniti toka skozi senzor. Na strani merilne opreme potrebujemo dodatno elektronsko vezje, ki tok skozi senzor pretvori nazaj v napetost, to merimo v nadaljnih stopnjah. Tokovna zanka omogoča tudi osnovno preverjanje senzorja, saj tok pod 4mA ali nad 20mA interpretiramo kot pokvarjen senzor.



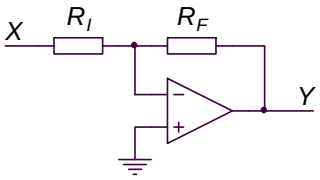
Slika 3.8: Tokovna zanka 4-20mA

3.2. Analogna obdelava signalov

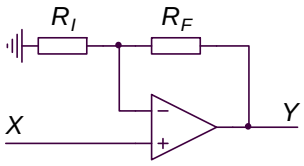
Kadar je opazovani analogni signal neprimeren za priključevanja na analogno digitalni pretvornik, ga analogno obdelamo. Pri tem mu lahko spremenimo velikost, ga prilagodimo na pretvornik ali spremenimo frekvenčni pas. Analogno obdelavo signala lahko uporabimo tudi za kompenzacijo znanih slabih lastnosti senzorja.

3.2.1. Operacijski ojačevalnik in ojačevalne stopnje

Elektronska vezja z operacijskimi ojačevalniki si lahko bralec ogleda v literaturi, tu omenimo le, da so ta vezja namenjena ojačenju in preoblikovanju signalov. Operacijski ojačevalnik je lahko vezan kot invertirani (slika 3.9, $Y = -X \cdot {}^{R_F}/_{R_I}$) ali neinvertirani ojačevalnik (slika 3.10, $Y = X \cdot (1 + {}^{R_F}/_{R_I})$). Pogosto je primernejši slednji, ker je izhodni signal enakega predznaka kot vhodni in je njegova vhodna upornost zaradi velike vhodne upornosti operacijskega ojačevalnika velika.



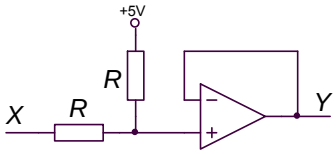
Slika 3.9: Operacijski ojačevalnik v invertirani konfiguraciji



Slika 3.10: Operacijski ojačevalnik v ne-invertirani konfiguraciji

Moderni operacijski ojačevalniki dopuščajo, da se vhodni in izhodni signali skoraj popolnoma približajo napajalnim napetostim (»rail-to-rail« izvedbe ojačevalnikov), zato lahko za napajanje teh stopenj uporabimo isto napetost, ki služi za napajanje pretvornika in procesorja. Pri tem je treba upoštevati, da se izhodni signal operacijskega ojačevalnika približa napajalni napetosti do nekaj 10mV. Če na primer napajamo operacijski ojačevalnik z napetostjo +5V in se merilno območje razteza od 0V do +5V, je začetek merilnega območja pri 0V in konec merilnega območja pri +5V popačen.

Kadar je vhodna napetost bipolarna (obseg je od – do +), sistem za pretvorbo iz analognega v digitalno pa je sposoben zajeti le pozitivne vhodne napetosti, moramo v tej stopnji vhodnemu signalu prišteti konstantno vrednost. Ta navadno znaša ½ obsega pretvornika. Uporabimo lahko vezje s slike 3.11, ki prevede obseg vhodnih napetosti od -5V do +5V v območje 0 do 5V tako, da ga stisne in prestavi za 2,5V. Vhodna upornost vezja je enaka 2*R*. V programu računalnika moramo ta analogni premik upoštevati.



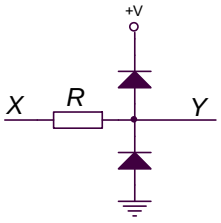
Slika 3.11: Tako enostavno stisnemo in premaknemo skalo

3.2.2. Vhodna impedanca ojačevalnika

Vhodna upornost ojačevalne stopnje je podana kot kvocient vhodne napetosti in vhodnega toka. Ker želimo z merjenjem čim manj motiti, naj bo vhodni tok čim manjši, torej vhodna impedanca čim večja. To dosežemo na primer z izbiro ojačevalne stopnje ne-invertiranega ojačevalnika ter operacijskega ojačevalnika z majhnim vhodnim tokom.

Kadar vodimo signal do merilnega sistema po oklepljenem kablu in se signal hitro spreminja, lahko pride do popačitev signala zaradi odbojev na koncu kabla. Odbojem se ognemo, če na koncu kabla priključimo upornik, katerega vrednost je enaka karakteristični upornosti kabla, ta je navadno 50Ω (75Ω). V tem primeru je seveda vhodna upornost ojačevalne stopnje in operacijskega ojačevalnika brezpredmetna, če je le za vsaj velikostni red večja od karakteristične upornosti kabla.

3.2.3. Območje vhodnih napetosti in vezja za zaščito

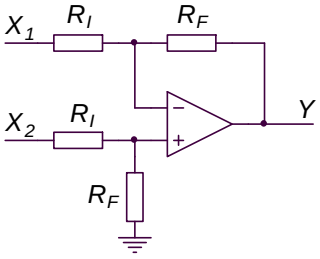


Slika 3.12: Diode zaščitijo pred prevelikimi vhodnimi napetostmi

Kadar vhodne napetosti presegajo napajalne napetosti vezja, lahko pride do poškodb elementov v vezju. Zato moramo poskrbeti, da do tega nikoli ne pride. Uporabimo lahko zaščitne diode po sliki 3.12. Pri tem +V predstavlja največjo dovoljeno vhodno napetost v pretvornik. Signal Y se lahko giblje le od -0,6V do +V+0,6V, če znaša padec napetosti na diodi 0,6V. Vrednost upornika izberemo glede na dopustni tok diod in vhodnega signala ter pričakovane največje vrednosti vhodnega signala.

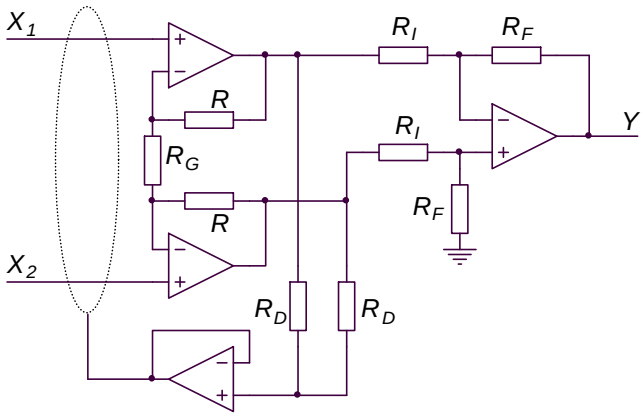
Sodobni vmesniki imajo zaščito pred prevelikimi vhodnimi napetostmi že vgrajeno, stopnja zaščite in največja dopustna vhodna napetost pa sta podana v specifikacijah naprave.

3.2.4. Diferenčni in instrumentacijski ojačevalnik



Slika 3.13: Diferenčni ojačevalnik

S senzorja dobimo napetost, ki je razlika dveh potencialov. Kadar noben od teh potencialov ni na nivoju ozemljitve merilnega sistema, uporabimo diferenčni (slika 3.13, $Y = (X_2 - X_1) \cdot {}^{R_F}/_{R_I}$) ali instrumentacijski ojačevalnik (slika 3.13, $Y = (X_2 - X_1) \cdot (1 + {}^{R}/_{2R_G}) \cdot {}^{R_F}/_{R_I}$). Pravimo, da priključimo senzorje na način RSE (»Referenced Single Ended«, kadar je referenčni potencial za vse v sistem vgrajene senzorje enak) ali DIFF



Slika 3.14: Instrumentacijski ojačevalnik
z vezjem za zaščito vhodnih signalov pred motnjami

(»Differential«, kadar senzorji nimajo skupnega referenčnega potenciala). Pri tem ima prednost shema 3.14, pri kateri sta tokova v oba vhodna priključka enaka in hkrati majhna, saj ju določa vhodni tok v operacijski ojačevalnik, ta pa je pri nekaterih operacijskih ojačevalnikih manjši od 10^{-15}A.

Spodnji operacijski ojačevalnik skupaj z delilnikom napetosti, ki ga tvorita upornika z oznako R_D služi računanji srednje vrednosti napetosti X_1 in X_2, ki jo uporabimo kot potencial za oklop kabla za signala X_1 in X_2.

3.2.5. Frekvenčni spekter signala in prepustni pas ojačevalnika

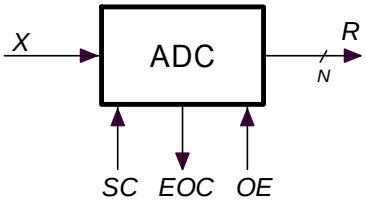
Opazovani signal prenaša koristno informacijo, ki obsega le del frekvenčnega spektra. Če gre na primer za avdio signal, je frekvenčni obseg omejen na frekvence med 20Hz in 20kHz. Ojačevati in obdelovati tisti del frekvenčnega spektra, ki ne vsebuje koristnih informacij, je nesmiselno, zato je frekvenčni spekter ojačevalnih stopenj navadno omejen. To hkrati izboljša razmerje med velikostjo koristnega signala in šuma, saj je šum porazdeljen po vsem frekvenčnem spektru, z omejevanjem frekvenčnega pasu ojačevalnika pa se efektivna vrednost šuma zmanjša.

3.3. Analogno digitalni pretvornik

Uvod: točka 1 velja za čipe, ostalo velja v vsakem primeru.

3.3.1. Signali

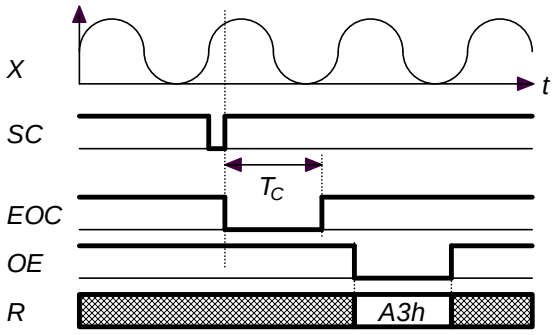
Na sliki 3.15 je bločna shema tipičnega analogno digitalnega pretvornika, ki je izdelan v obliki integriranega vezja. Na vhod je priključen merjeni signal *X*, delovanje pretvornika pa nadzirajo trije digitalni signali *SC* (»Start Conversion«, začni s pretvorbo), *EOC* (»End Of Conversion«, pretvorba opravljena) in *OE* (»output enable«, omogoči izhod). Rezultat pretvorbe je na razpolago na vodilu *R*, ki ga sestavlja *N* bitov.



Slika 3.15: Tipični analogno-digitalni pretvornik in signali

Potek signalov za tak pretvornik je na sliki 3.16. Pretvornik čaka na ukaz za začetek konverzije, ki ga predstavlja rastoči rob signala *SC*. Takrat si pretvornik zapomni analogno vrednost ter jo začne pretvarjati v digitalni ekvivalent, hkrati pa spremeni vrednost signala *EOC* ter tako potrdi delovanje. Za pretvorbo potrebuje nekaj časa (glej Čas konverzije, 3.3.6). Ko je pretvorba končana, je rezultat shranjen v pretvorniku, signal *EOC* pa dobi prvotno vrednost in šele takrat lahko rezultat preberemo in uporabimo. V ta namen je pretvornik opremljen s priključkom za signal *OE*. Kadar ima signal *OE* vrednost nič, je rezultat na voljo na vodilu *R*.

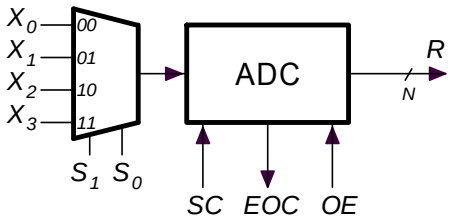
Na razpolago so tudi analogno-digitalni pretvorniki, kjer je rezultat pretvorbe dostopen preko serijskega vodila. V tem primeru potrebujemo procesorski sistem z ustreznim vodilom.



Slika 3.16: Signali za ADC

3.3.2. Število vhodnih priključkov

Navadni analogno digitalni pretvornik ima le en priključek za merjeno napetost. Bolj kompleksni pretvorniki imajo vgrajen analogni multiplekser, ki ima več priključkov za vhodne napetosti, slika 3.17. S kombinacijo vrednosti kontrolnih signalov S_x za multiplekser se odločimo, katero od vhodnih napetosti bomo pomerili. Naenkrat lahko merimo le eno vhodno napetost. Kadar izmenoma merimo več vhodnih signalov, izmerki niso sočasni. To v nekaterih primerih ne moti ali pa lahko kompenziramo v programski opremi. Kadar to ni mogoče, potrebujemo več ločenih pretvornikov, za vsak signal po enega.



Slika 3.17: Z multiplekserjem povečamo število vhodnih signalov v ADC

3.3.3. Obseg vhodnih napetosti in vhodna upornost

Obseg vhodnih napetosti, ki jih pretvornik uspešno pretvori v digitalni ekvivalent, je omejen in ga podaja proizvajalec. Tipične vrednosti znašajo $\pm 1\mathrm{V}$, $\pm 10\mathrm{V}$, $0\mathrm{V}..5\mathrm{V}$ in podobno. Če na vhod pretvornika priključimo napetost, ki obseg vhodnih vrednosti presega, lahko pretvornik trajno poškodujemo. Večina pretvornikov je proti takim poškodbam elektronsko zaščitena, vendar le, če prekoračitev ni prevelika.

Rezultat *REZ* meritve je številka, ki je med 0 in $2^N - 1$ za unipolarne pretvornike oziroma med -2^{N-1} in $2^{N-1} - 1$ za bipolarne pretvornike, pri tem je *N* število bitov pretvornika (glej »ločljivost, 3.3.4«). Za unipolarne pretvornike je obseg vhodnih napetosti U_X med 0 in (skoraj) U_{REF}:

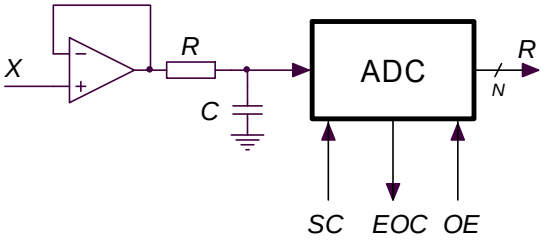
$$REZ = \frac{U_X}{U_{REF}} \cdot 2^N$$

Za bipolarne pretvornike pa med $-U_{REF}$ in (skoraj) $+U_{REF}$:

$$REZ = \frac{U_X}{U_{REF}} \cdot 2^{N-1}$$

Vhodna upornost pretvornika je definirana kot kvocient vhodne napetosti in vhodnega toka vanj. Želimo, da bi bila vhodna upornost čim večja in zato vhodni tok čim manjši, ker tak pretvornik manj vpliva na vir signala. Vhodni tok v večino pretvornikov je večji med merjenjem in med preklapljanjem multiplekserja, zato je smiselno med vir signala in pretvornik vstaviti enoto z ojačenjem ena, ki tak tok zmore. Pogosto si pomagamo tudi z RC členom po sliki 3.18, ki zaradi velike kapacitivnosti kondenzatorja *C* (tipično vsaj 10nF) pokriva potrebe po toku analogno digitalnega pretvornika med merjenjem. Upornik R izberemo glede na zgornjo frekvenco, ki naj bi jo RC člen še ne prizadel.

Če je vir signala šibek in tokovnega ojačevalnika ne moremo vstaviti, je smiselno po preklopu multiplekserja vsaj počakati, da prehodni pojavi zaradi povečanega toka ob preklopu izzvenijo.



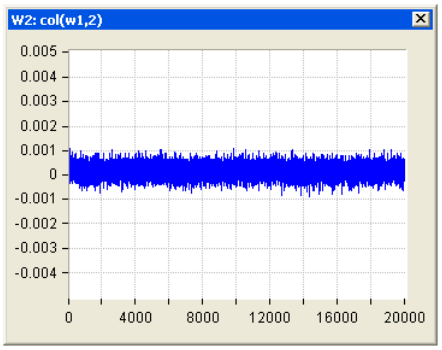
Slika 3.18: Tako izničimo vpliv povečanega vhodnega toka ADC-ja med pretvorbo

3.3.4. Ločljivost (»resolution«) in efektivno število bitov (ENOB, »Effective Number Of Bits«)

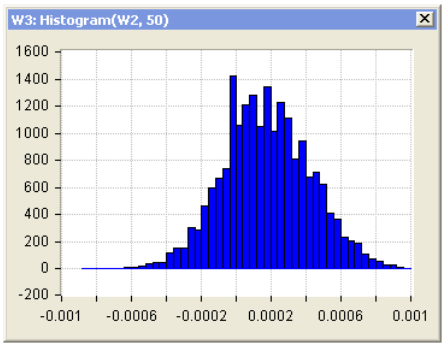
Ločljivost je definirana kot število bitov v rezultatu pretvornika; če izhodno vodilo pretvornika vsebuje osem žic, je torej ločljivost osem bitov. Alternativno lahko navedemo kot ločljivost tudi število diskretnih vrednosti, ki jih lahko dobimo na izhodu pretvornika, za osem bitni pretvornik je to 256.

Na videz lahko ločljivost pretvornika večamo v nedogled tako, da dodajamo bite izhodnemu signalu. Tako definirane ločljivosti ne gre pomešati z dejansko ločljivostjo pretvornika, ki jo podaja efektivno število bitov ENOB (»Effective Number of Bits«). Zaradi nepopolnosti izdelave pretvornika in šumov v merilnem sistemu se izmerki sicer konstantnega vhodnega signala raztresejo. Efektivno število bitov takrat določimo tako, da odčitke razvrstimo v histogram in odčitamo ali izračunamo njihovo standardno deviacijo σ. Efektivna ločljivost pretvornika ENOB je potem enaka razliki med ločljivostjo pretvornika v bitih in standardno deviacijo σ.

Zgled: Na sliki 3.19 je diagram zajetega signala. Uporabljen je bil osciloskop z osem bitnim ADC-jem in vhodno občutljivostjo ±5mV. Histogram zajetega signala je na sliki 3.20, od tu odčitamo $\sigma = 0{,}2538\text{mV} \rightarrow 2{,}7$ bita. Dejanska ločljivost ENOB merilnega sistema je torej $8 - 2{,}7 = 5{,}3$ bita.



Slika 3.19: Zajeti signal, horizontalna os je brezdimenzijska, vertikalna v mV



Slika 3.20: Histogram zajetega signala, horizontalna os je v mV

3.3.5. Točnost

Točnost analogno digitalnega pretvornika je enaka razmerju:

$$T = \frac{M - R}{R}$$

V zgornjem izrazu *T* predstavlja točnost, *M* izmerjeno in *R* dejansko vrednost. Točnost je odvisna od zgradbe pretvornika in točnosti referenčnih elementov, tipične vrednosti pa delci odstotka.

3.3.6. Čas konverzije

Analogno digitalni pretvornik potrebuje nekaj časa, da določi pravilni digitalni ekvivalent vhodne napetosti. Ta čas imenujemo čas konverzije T_C, glej sliko 3.16. Čas konverzije je odvisen od principa, ki je uporabljen za konverzijo, ločljivosti in kakovosti uporabljenih vezij. Tipične vrednosti se gibljejo od nanosekund pri zelo hitrih ADCjih z ločljivostjo največ osem bitov do desetin milisekund pri pretvornikih visoke ločljivosti.

3.3.7. Napaka zaradi časa vzorčenja

Pričakujemo vzorčenje vhodnega signala v enakih časovnih intervalih. Če intervali niso enaki, pretvornik zaradi tega v napačnem trenutku pomeri vhodno napetost in na videz napačno določi njeno velikost. Vzemimo, da vzorčimo harmonično napetost, katere velikost je enaka obsegu vhodnih napetosti pretvornika. Velikost bo pravilno določena, če je napaka zaradi nepravega časa vzorčenja manjša od en LSB. Velja formula:

$$\Delta t = \frac{T}{\pi \cdot 2^N}$$

Kjer je *T* perioda opazovanega harmonskega signala, *N* ločljivost pretvornika v bitih in $\triangle t$ največje dovoljeno odstopanje od pravega trenutka vzorčenja. Tabela 3.21 podaja nekaj značilnih vrednosti.

Perioda pojava	8 pretvornik	12 bitni pretvornik	16 bitni pretvornik
1s	1,2ms	77μs	4,8μs
1ms	1,2μs	77ns	4,8ns
1μs	1,2ns	77ps	4,8ps

Tabela 3.21: Časovna netočnost, ki dopušča napako, manjšo od 1LSB

3.3.8. Integralna in diferencialna nelinearnost

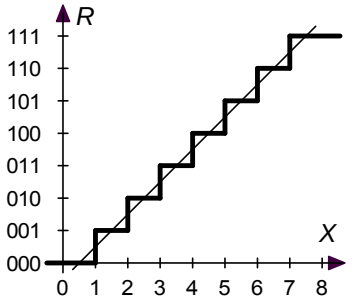
V idealno linearnem pretvorniku slika 3.21 predstavlja relacijo med zvezno spreminjajočo vhodno napetostjo *X* in izhodnim diskretnim rezultatom *R*. Pri realnem pretvorniku je relacija slabša, zgled je na sliki 3.21. Vse stopničke niso enako široke, kar je posledica slabe izdelave pretvornika. Pri današnjih pretvornikih proizvajalci večinoma garantirajo sledeče.

- Pri povečevanju analogne vhodne napetosti dobimo vsakokrat večjo izhodno digitalno vrednost, lastnost se imenuje monotonost. Za večjo vhodno napetost ne moremo dobiti manjše izhodne vrednosti.
- Širina stopničke, ki bi morala znašati 1LSB (»Least Significant Bit«), se lahko giblje od nič do 2LSB, znaša torej LSB $\pm$1LSB. Vrednost LSB določimo iz računa:

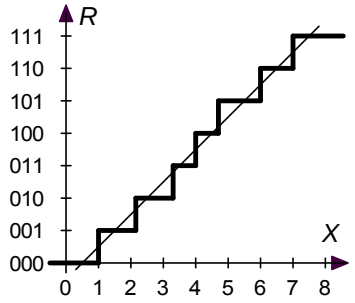
$$\text{LSB} = \frac{obseg\ vhodnih\ napetosti}{2^{ločljivost}}$$

- Boljši pretvorniki imajo širino stopničke definirano kot LSB $\pm\,{}^{1}/_{2}$LSB.

Lastnost, zaradi katere niso vse stopničke enako široke, označimo z imenom diferencialna nelinearnost in jo za ovrednotimo kot širino tiste stopničke, ki najbolj odstopa od nominalne vrednosti LSB.



Slika 3.20: Relacija med izhodom in vhodom za idealni ADC



Slika 3.20: Relacija med izhodom in vhodom za realni ADC

Ker ni nujno, da je posamezna stopnička širša na račun sosednjih stopničk, lahko na primer več zaporednih preširokih stopničk pokvari tudi točnost meritve. Ker je ta vrsta nelinearnosti povezana z vsoto širin posameznih stopničk, jo imenujemo

integralna nelinearnost. Posledica integralne nelinearnosti je nesorazmernost namerjene z vhodno vrednostjo zaradi neenakih širin stopničk.

3.4. Procesor

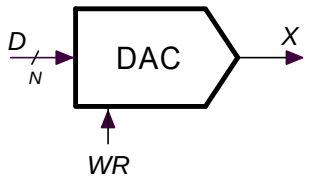
Je podrobno opisan v posebnem poglavju.

3.5. Digitalno analogni pretvornik

Digitalno analogni pretvornik je enota, ki sprejme digitalni, diskretni signal in ga pretvori v analognega, zveznega. Kadar govorimo o integriranih vezjih, je izhodna vrednost pretvornika tok, ki ga s pomočjo operacijskega ojačevalnika spremenimo v napetost. Pogosto je operacijski ojačevalnik že vgrajen v integrirano vezje, zato je izhodni signal napetost, z zunanjimi elementi pa izberemo le še območje izhodnih napetosti.

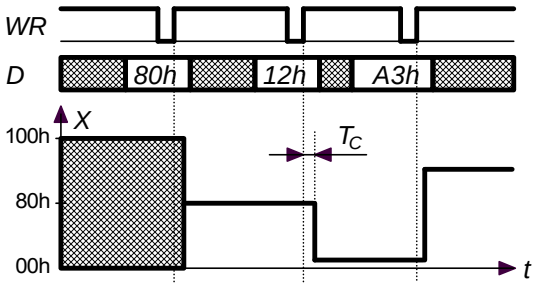
3.5.1. Signali

Digitalno analogni pretvornik v obliki integriranega vezja je narisan kot blok na sliki 3.21. Ima niz digitalnih priključkov za vhodno digitalno vrednost, ta niz je izveden kot vodilo *D*, vsaj en digitalni kontrolni signal *WR*, ki definira čas vpisovanja digitalne vrednosti v pretvornik in vsaj en analogni izhodni signal U_o. Na vodilo *D* najprej postavimo digitalno vrednost, ki jo želimo pretvoriti v analogni ekvivalent, nato signal *WR* iz vrednosti ena spremenimo v nič in nazaj v ena, slika 3.22. Tako vpišemo digitalno vrednost v pretvornik in ta se odzove z ustrezno analogno vrednostjo na izhodu. Če je izhodna vrednost napetost, ki je omejena z U_L in U_H ter *N* število bitov pretvornika, je izhodna napetost U_o podana z:



Slika 3.21: Simbol za DAC

$$U_o = \frac{D}{2^N} \cdot (U_H - U_L) + U_L$$



Slika 3.22: Potek signalov za vpisovanje v DAC

3.5.2. Obseg izhodnih napetosti ali tokov

Izhodna napetost in največji dovoljeni izhodni tok je podan v specifikacijah proizvajalca. Tipične vrednosti za U_L in U_H so -10V do +10V ali 0V do +5V, največji izhodni tok pa je navadno omejen na vsega nekaj mA. Kadar potrebujemo večji izhodni tok ali napetost, uporabimo vezja za ojačenje signalov.

3.5.3. Zaščita

Digitalno analogni pretvornik daje signal v specificiranem območju, pri tem je na voljo omejena količina toka. Če na izhod pretvornika priključimo drug vir signala, ki skuša signal z DAc-a po svoje spremeniti, bo en od virov signala popustil in morda odpovedal. Težavam se ognemo na trivialni način: na izhodne sponke digitalno analognega pretvornika priključujemo le porabnike, ki ustrezajo specifikacijam proizvajalca pretvornika. Izhodne sponke pretvornika lahko pred prevelikim tokom zaščitimo tako, da zaporedno z izhodom priključimo upornik, vendar na ta način spremenimo lastnosti pretvornika.

3.5.4. Ločljivost in efektivno število bitov

Ločljivost (resolucijo) digitalno analognega pretvornika spet definiramo kot število bitov v vhodnem digitalnem vodilu *D*. Število bitov v vhodnem digitalnem vodilu lahko proizvajalci integriranih vezij poljubno povečujejo, pa to ne prispeva nujno h kvaliteti pretvornika. Zato pogosto podajamo še efektivno število bitov ENOB, ki je definirano enako kot za analogno digitalni pretvornik.

3.5.5. Točnost

Velja isto, kot za analogno digitalni pretvornik.

3.5.6. Čas konverzije

Od trenutka, ko vpišemo novo digitalno vrednost v pretvornik do trenutka, ko se ustrezen analogni ekvivalent pojavi in ustali na izhodu pretvornika, mine nekaj časa. Ta čas imenujemo čas konverzije T_C in tipično znaša do nekaj mikrosekund, glej sliko 3.22. Čas konverzije je odvisen od velikosti spremembe izhodnega signala *X*, zato proizvajalci podajajo najdaljši možni čas konverzije.

3.6. Analogna izhodna vezja

3.6.1. Ojačenje in obseg izhodnih napetosti ter tokov

Ojačenje in obseg izhodnih signalov mora biti prilagojeno bremenom, ki so na vezja priključena. Pri tem je pogosto v analogno izhodno vezje vgrajena še enota, ki preprečuje poškodbe zaradi napačne priključitve bremena (kratek stik, odprte sponke, reaktivno breme, …). Ta enota omeji izhodni tok in napetost na varno vrednost. Zdi se razumno, da se na delovanje zaščitne enote vseeno ne zanašamo, ampak skušamo preprečiti napačne priključitve.

3.6.2. Izhodna impedanca

Izhodna impedanca je prilagojena bremenu, ki je na vezje priključeno. Kadar poskušamo z univerzalnimi vezji, je izhodna impedanca tipično 50Ω.

3.6.3. Frekvenčna karakteristika

Frekvenčna karakteristika analognih izhodni vezij je prilagojena frekvenčnemu pasu generiranega signala. V tej stopnji tudi pogosto opravimo filtriranje generiranega signala, ki navadno vsebuje frekvenčne komponente, enake frekvenci pisanja nove digitalne vrednosti v pretvornik.

3.7. Aktuatorji

Aktuatorji so elementi, ki električne signale pretvorijo v fizikalne veličine. Mednje štejemo motorje, grelnike, svetila, črpalke, … Vsi si elementi potrebujejo za svoje delovanje več električne energije, kot jo lahko preskrbi običajna digitalna ali analogna elektronika. Poganjati jih bo torej treba preko njim namenjenih ojačevalnikov ali prilagodilnih členov.

3.7.1. Električne lastnosti

Električne lastnosti aktuatorjev podaja proizvajalec in jih tu ne bomo podrobneje navajali, saj je obseg vrednosti zelo velik.

3.7.2. Napajanje in ozemljitve

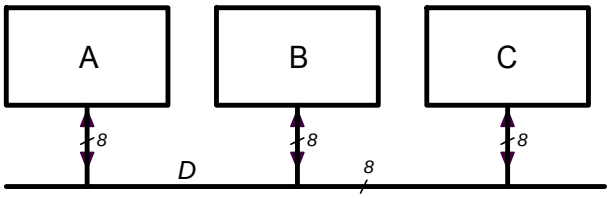
Aktuatorji potrebujejo za delovanje več električne moči, zato je njihovo napajanje smiselno galvansko ločiti od napajanja procesorskega sistema in občutljivih analognih vezij. V ta namen uporabljamo releje ali še bolje optične sklopnike tako, da so močnostni deli elektronike popolnoma oddvojeni od občutljivejših malih napajalnih napetosti za analogno elektroniko in procesor.

4. Vodila v digitalnih sistemih

Vodilo je niz žic, ki je namenjen medsebojni izmenjavi podatkov med v sistem povezanimi enotami. Enot je navadno več, podatke med njimi pa želimo izmenjevati čim hitreje po čim manjšem številu žic. Zahtevi se med sabo izključujeta, zato racionaliziramo vodila v paralelna, ki omogočajo velike hitrosti izmenjave podatkov ter serijska, ki shajajo z majhnim številom žic, a je hitrost prenosa zato manjša. Signali na vodilu in protokol prenosa podatkov so definirani za vsako vrsto vodila posebej in temu morajo biti prilagojene tudi enote, ki jih na vodilo priključujemo.

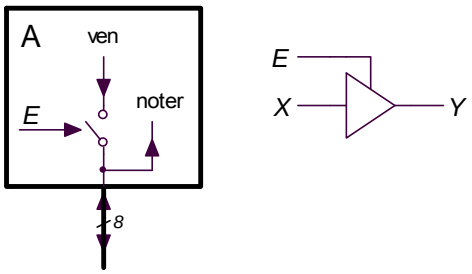
4.1. Paralelno vodilo

Vzemimo, da imamo opravka s procesorskim sistemom, ki je sestavljen iz treh enot A, B in C. Če želimo bajt podatka naenkrat poslati od enote A k enoti B, potrebujemo med enotama osem žic, za vsak bit po eno. Podobno potrebujemo osem žic med enotama A in C takrat, ko želimo drug bajt podatka poslati od enote A k enoti C. Iste žice lahko uporabimo tudi takrat, ko želimo en bajt podatka iz enote C prebrati v enoto A. Zato vse tri enote med sabo povežimo s podatkovnim vodilom po sliki 4.1, ki ga sestavlja osem žic; podatkovno vodilo je predstavljeno z debelejšo črto in oznako za število žic. Dogovorimo se še, da je možno naenkrat prenašati podatke le med enoto A (tipično procesorjem) in eno od ostalih enot; enota A ima poseben status in je vedno vključena v prenos podatkov.



Slika 4.1: Enote povežemo z vodilom D, ki je skupek osmih žic

Na podatkovno vodilo lahko istočasno le ena enota pošilja podatke, zato so vse enote opremljene s tako imenovanim tri-stanjskim izhodom. Enota lahko na podatkovno vodilo pošilja podatke kadar je stikalo na sliki 4.2 sklenjeno (dejansko osem stikal, na sliki je zaradi preglednosti narisano le eno), takrat lahko na

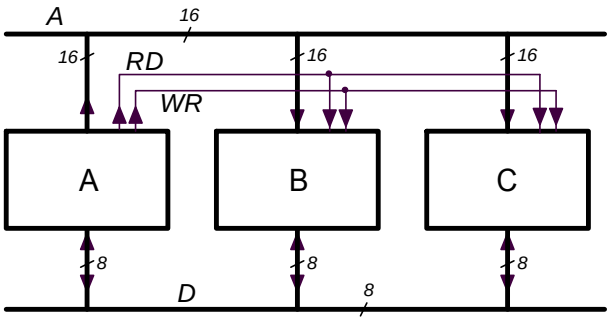


Slika 4.2: Notranjost enote, ki omogoča priključitev na dvosmerno vodilo

podatkovno vodilo pošilja bodisi enke ali ničle. Kadar je stikalo razklenjeno, enota podatkov na vodilo ne more pošiljati; pravimo, da je enota v stanju visoke impedance 'Z'. Stikalo je elektronsko, krmili ga signal *E* (»enable«, slika 4.2 desno).

S podatkovnega vodila lahko katerakoli enota odjema podatke. Ker so podatki za izbrano enoto prisotni na vodilu le kratek čas, mora vsaka enota vsebovati register, vanj se vpiše podatek z vodila in ostane na razpolago za kasneje.

Enota A (procesor) je tista enota, ki nadzira smer in čas prenosa podatkov. Zato daje še dva kontrolna signala, s katerima ostalim enotam sporoča smer prenosa podatkov: od ostalih enot proti procesorju (»read«, *RD*) ali od procesorja proti ostalim enotam (»write«, *WR*). Procesor tudi ve, s katero enoto želi izmenjevati podatke, zato je opremljen še z naslovnim vodilom. Preko naslovnega vodila procesor pošilja kodo tiste enote, s katero želi trenutno izmenjevati podatke. Ker je enot v procesorskem sistemu lahko veliko, naslovno vodilo sestavlja veliko žic. Tipična vrednost je 16, tak procesor lahko naslovi do 2^{16} = 65536 enot. Bločna shema procesorskega sistema z obema vodiloma in kontrolnima linijama je na sliki 4.3. Ostale na podatkovno vodilo priključene enote morajo dekodirati signale z naslovnega vodila in kontrolnih linij ter se ob praven trenutku odzvati na ukaze enote A (procesorja).



Slika 4.3: Vodilo dopolnimo z linijami A RD in WR, ki omogočajo glavni enoti A (procesorju), da izbere ene od enot na vodilu in z njo izmenja podatke

Na sliki 4.4 levo so narisani signali na vodilih in kontrolnih linijah med branjem podatka iz enote B v enoto A (procesor). Pri tem je privzeto, da je enota B prirejena

tako, da se odzove na kodo 1000_H na naslovnem vodilu, enota C pa na kodo 2000_H. Na časovne intervale razčlenjeno branje je sestavljeno iz:

R1) Vrednost naslovnega in podatkovnega vodila ni definirana ali vsaj ni pomembna za ta zgled, vrednosti signalov *RD* in *WR* sta ena in torej neaktivni: procesor ne želi brati ali pisati.

R2) Procesor pošlje na naslovno vodilo kodo tiste enote (1000_H), s katero želi izmenjevati podatke. Vse enote dekodirajo naslovno vodilo in enota B prepozna vrednost z naslovnega vodila ter se pripravi na izmenjavo podatkov.

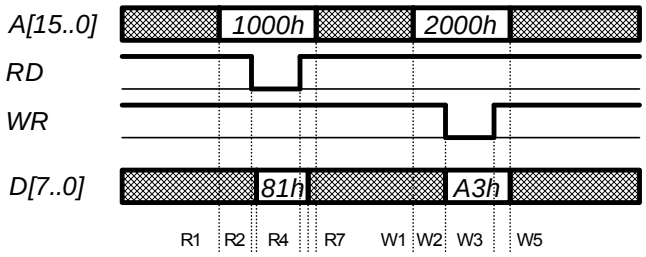
R3) Procesor aktivira signal *RD* (vrednost se spremeni na nič), s tem izbrani enoto B sporoči, da želi brati.

R4) Enota B se odzove tako, da aktivira svoj *E* signal, sklene stikala in pošlje podatke (81h) na podatkovno vodilo.

R5) Procesor podatke sprejme v tistem trenutku, ko deaktivira (vrne na ena) signal *RD*; prejete podatke shrani v enega od svojih registrov.

R6) Enota B preneha pošiljati podatke na podatkovno vodilo, saj signal *RD* ni več aktiven.

R7) Procesor spremeni vrednost na naslovnem vodilu, ki zdaj za ta zgled ni več pomembna.



Slika 4.4: potek signalov na vodilu

Hitrost spreminjanja signalov na podatkovnem vodilu je odvisna od hitrosti vseh na vodilo priključenih enot. Opisano dogajanje traja na vodilu enostavnega procesorkega sistema do nekaj mikrosekund, na vodilu v osebnem računalniku pa nekaj deset nanosekund.

Na sliki 4.4 desno so narisani signali pri pisanju enote A (procesorja) v enoto C, ki se odziva na naslov 2000_H. Dogajanje razčlenimo na:

W1) vrednost naslovnega in podatkovnega vodila ni definirana ali vsaj ni pomembna za ta zgled.

W2) Procesor pošlje na naslovno vodilo kodo tiste enote (2000_H), s katero želi izmenjevati podatke. Vse enote dekodirajo naslovno vodilo in enota C

prepozna vrednost z naslovnega vodila ter se pripravi na izmenjavo podatkov.

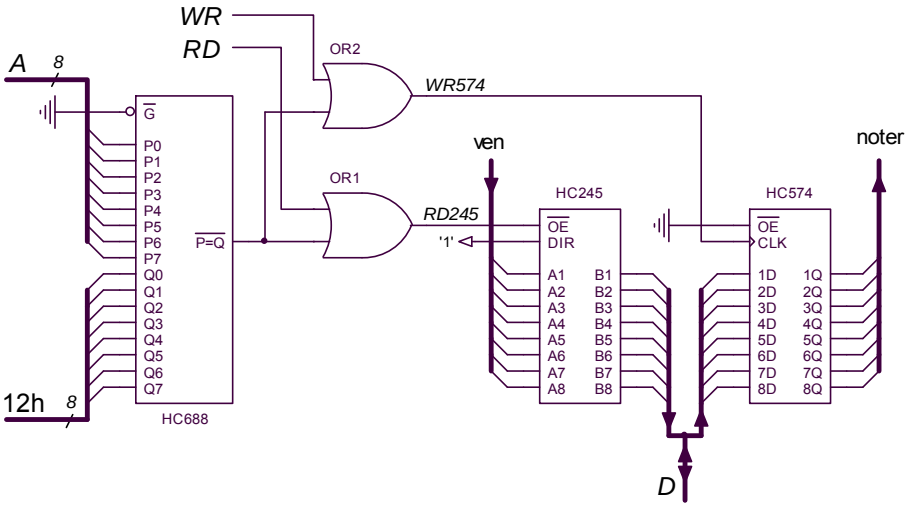
W3) Procesor pošlje podatek, ki ga želi vpisati v enoto C (na primer $A3_H$), na podatkovno vodilo ter hkrati aktivira signal *WR* (vrednost se spremeni na nič), s tem izbrani enoti C sporoči, naj se pripravi na pisanje.

W4) Procesor vrne vrednost signala *WR* v neaktivno stanje, preskok signala *WR* iz ničle v enko je sporočilo enoto C, da shrani vrednost s podatkovnega vodila v svoj register

W5) Procesor spremeni vrednost podatkovnega in naslovnega vodila, obe vrednosti nista več pomembni za ta zgled.

Čeprav je korakov, potrebnih za prenos podatka iz enote A v enoto C, manj, tipično trajajo enako dolgo kot branje iz enote.

Na sliki 4.5 je elektronska shema vezja, ki omogoča naslovno dekodiranje in priključitev enote na vodilo. Zaradi enostavnosti zgleda je uporabljeno naslovno vodilo z osmimi žicami, enota pa se odziva na naslov 12_H. Naslovno dekodiranje opravlja digitalni komparator HC688, njegov izhodni signal ima vrednost nič takrat, ko je na naslovnem vodilu *A* vrednost 12_H. Vrata OR1 generirajo signal *RD254* za pošiljanje vrednosti *'ven'* skozi tristanjski vmesnik HC245 iz enote na vodilo *D*, vrata OR2 pa signal *WR574*, ki omogoči shranjevanje podatka z vodila v register HC574.



Slika 4.5: Elektronska shema vezja, ki omogoča odjem in pošiljanje podatkov na paralelno vodilo

Od števila žic v naslovnem vodilu je odvisno, koliko različnih enot lahko naslovi procesor. Govorimo o pojmu naslovnega prostora, katerega velikost je 2^N, pri tem je *N* število žic naslovnega vodila. Šestnajst žic se na prvi pogled zdi veliko, saj je naslovni prostor velik 65536 mest. Ker vsaka spominska celica zaseda eno mesto v

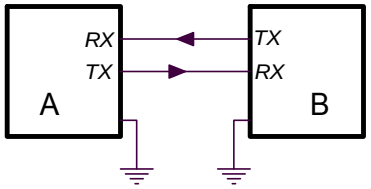
spominskem prostoru, to pri večjih računskih sistemih ne zadošča. Tam so naslovna vodila širša (24 ali 32 žic).

Zdi se nesmiselno, da naslovno dekodiranje ločeno sestavljamo za vsako enoto posebej. Temu se ognemo, če sestavimo enotno naslovno dekodiranje, do posameznih enot pa vodimo le signale za aktivacijo.

4.2. Serijska vodila

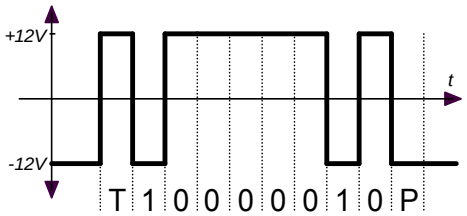
4.2.1. RS232

Vodilo RS232 je namenjeno dvosmerni izmenjavi podatkov med dvema računalnikoma. Za osnovno verzijo vodila potrebujemo vsega tri žice po sliki 4.6. Prva žica je namenjena pošiljanju podatkov od enote A k enoti B, druga je namenjena pošiljanju podatkov v obratni smeri. Tretja žica je referenčna ničla. Vsaka od enot je opremljena z oddajnikom in sprejemnikom, ki generirata vse potrebne signale. Oddajnik pošilja podatke zaporedno bit za bitom, sprejemnik mora prejeti niz bitov spet sestaviti v prvotno obliko. Vodilo je asinhrono, zato mora sprejemnik v naprej poznati hitrost pošiljanja podatkov na strani oddajnika, poleg tega mora sprejemnik brez dodatne pomoči iz poslanega niza vrednosti izluščiti poslane podatke.



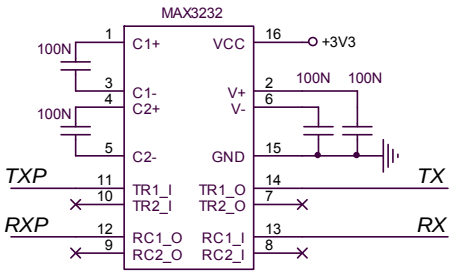
Slika 4.6: Povezava RS232 med enotama

Na sliki 4.7 je narisan tipičen signal, ki ga oddajnik pošlje za črko 'A' (ASCII koda 41_H). Nizu osmih bitov sta dodana tako imenovan »start bit« T z vrednostjo nič, ki označuje začetek niza in »stop bit« P z vrednostjo ena, ki označuje njegov konec. Ta dva bita omogočata sinhronizacijo sprejemnika z oddajnikom. Po specifikaciji RS232 standarda mora oddajnik poslati bit ena kot napetost, bolj negativno od -9V, ničlo pa kot napetost, večjo od +9V. Najprej se pošilja LSB, nazadnje MSB. Zaradi večje zanesljivosti prenosa podatkov mora biti sprejemnik sposoben napetost nad približno 2V interpretirati kot logično nič, napetost pod 0V pa kot logično ena.



Slika 4.7: Signal RS232 za črko 'A'

Standardne hitrosti prenosa podatkov so na primer 9600 bitov na sekundo (BPS, »Bit Per Second«, Baud, Bd), 19200Bd, 38400Bd, 115200Bd in naprej. Hitrost prenosa podatkov na oddajni strani mora biti enaka nastavljeni hitrosti prenosa podatkov na sprejemni strani, izbira hitrosti pa je odvisna od sposobnosti uporabljenih naprav in dolžine ter kakovosti povezave med njimi.



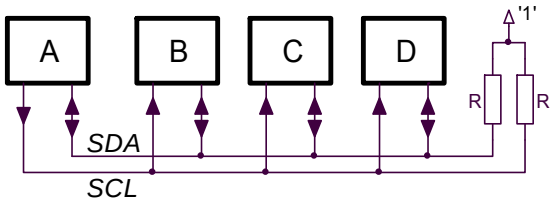
Slika 4.8: Vezje za prilagoditev nivojev in TTL v RS232 standard. TXP in RXP sta priključka na procesor, TX in RX sta liniji do druge enote

Napetostni nivoji RS232 vodila so nestandardni za logična vezja, zato navadno uporabljamo napetostne pretvornike, ki signala TXD in RXD prilagodita na običajne logične nivoje uporabljenih integriranih vezij. Tipično vezje za prilagoditev nivojev je na sliki 4.8. Kadar se tak protokol prenosa podatkov uporablja med integriranimi vezji na isti plošči tiskanega vezja, lahko vezja za prilagoditev nivojev opustimo ter delamo s signali, ki so po napetostih kompatibilni z uporabljenimi logičnimi vezji.

Za povečanje zanesljivosti prenosa podatkov se nizu včasih pritakne še dodatni »bit parnosti«, na podlagi vrednosti tega bita sprejemnik laže odkrije morebitne napake pri prenosu. Prav tako je zaradi povečanja zanesljivosti prenosa osnovnim trem žicam lahko dodanih še več, preko teh žic enoti sporočata svojo zasedenost oziroma pripravljenost za sprejemanje ter oddajanje. Pri enostavnih prenosih dodatni biti in signali niso potrebni in jih lahko opustimo.

4.2.2. Vodilo I²C

Vodilo I^{2}C (IIC, »Inter Integrated Circuit«) je namenjeno izmenjavi podatkov med dvema ali več integriranimi vezji (IC) na nivoju enega tiskanega vezja. Glavni razlog za njegovo vpeljavo je bilo zmanjšanje potrebnih žic za povezovanje integriranih vezij.



Slika 4.9: Povezava med enotami na I^{2}C vodilu

Vodilo tvorita dve žici *SCL* in *SDA*. Referenčna ničla navadno ni posebej navedena, ker so ICji na nivoju istega tiskanega vezja že povezani na skupno ozemljitev. Eno od integriranih vezij, navadno mikroprocesor (A), ima poseben status in nadzoruje pretok podatkov po vodilu, imenujemo ga »master«. Ostala integrirana vezja (B, C, D, »slave«) se odzivajo na ukaze glavne enote. Vsi prenosi podatkov se vršijo med vezjem »master« in enem od vezij »slave«, nikoli direktno med vezji »slave«.

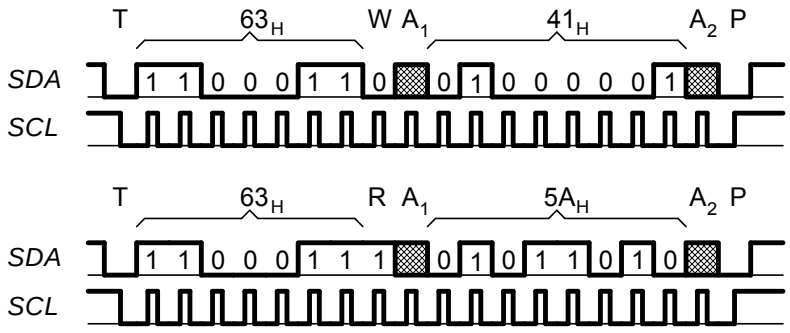
Prenos podatkov je sinhron. Vezje »master« oddaja signal ure *SCL*, s katerim sinhronizira delovanje vseh na vodilo priključenih vezij. Signal *SDA* služi dejanskemu prenašanju podatkov. Signal *SDA* se sme spremeniti le takrat, ko ima signal *CLK* vrednost nič.

Vrednosti napetosti so le ohlapno definirane, hitrost prenosa podatkov je le navzgor omejena s sposobnostjo vezij. Obe liniji sta preko upornikov R stalno vezani na logično ena, posamezno vezje lahko linijo kratko veže na nič. Tako se ognemo poškodbam vezij takrat, ko bi več vezij hkrati skušalo pošiljati podatke na linije.

Na istem vodilu je več vezij, zato je prenos bajta podatkov sestavljen iz več delov.

- »Start kombinacija« (T): ta kombinacija vrednosti signalov *SCL* in *SDA* vsem vezjem na vodilu sporoči, da se prenos podatkov začenja; obe liniji poganja »master«, *SCL* je enak ena, *SDA* se spremeni iz enke v ničlo, nato se *SCL* spremeni v ničlo.
- »Naslov«: ker je vezij več, »master« najprej po vodilu sporoči naslov tistega od »slave« vezij, s katerim želi komunicirati; naslov je sestavljen iz sedmih bitov, osmi bit v nizu pa predstavlja smer prenosa podatkov (nič za pisanje, ena za branje). Obstaja tudi izpopolnjena verzija I^2C protokola, kjer je naslavljanje devet-bitno.
- »Prva potrditev« (A_1): vezje, ki prepozna svoj naslov in je pripravljeno na komunikacijo, potrdi svojo pripravljenost tako, da za en sunek signala *SCL* potegne linijo *SDA* na vrednost nič.
- »Podatki«: vezje, ki je naslovljeno, v tem intervalu pošilja ali sprejema podatke z linije *SDA* v ritmu, ki ga diktira linija *SCL*; podatkovnih bitov je osem.
- »Druga potrditev« (A_2): vezje, ki je v prejšnjem koraku sprejemalo podatke, za en sunek ure pošlje na linijo *SDA* vrednost nič in tako potrdi, da je pravilno sprejela podatke.
- »Stop kombinacija« (P): ta kombinacija vrednosti signalov *SCL* in *SDA* vsem vezjem na vodilu sporoči, da je prenos podatkov za zdaj končan; *SDA* je ničla, *SCL* se spremeni iz ničle v enko, nato se v enko spremeni še *SDA*.

Za zgled naj služi zgornja polovica slike 4.10, kjer vezje »master« najprej naslovi vezje 63_H, prejme od njega potrditev, nato pa temu vezju pošlje podatek 'A' (41_H), kar naslovljeno vezje ponovno potrdi. Podobno poteka tudi branje podatka iz istega vezja (isti naslov, drugačen zadnji bit v nizu, preberemo vrednost $5A_H$), kar je prikazano na isti sliki spodaj.

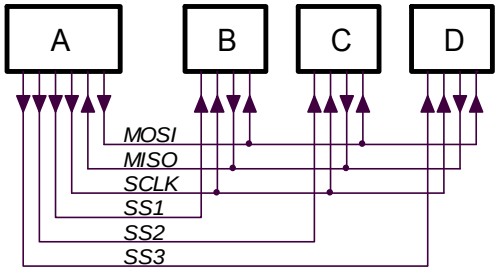


Slika 4.10: Signali na vodilu I2C: zgoraj pisanje, spodaj branje v procesor

4.2.3. Vodilo SPI

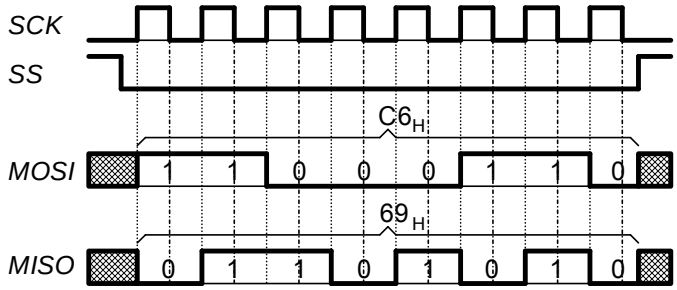
Vodilo SPI (»Serial Peripheral Interface«) je namenjeno pošiljanju podatkov med dvema ali več integriranimi vezji na isti plošči tiskanega vezja. Tudi tu je glavni razlog za vpeljavo zmanjšanje števila povezovalnih žic.

Vodilo sestavljajo štiri žice s signali *SCLK* (»Serial Clock«), *MOSI* (»Master Out & Slave Input«), *MISO* (»Master Input & Slave output«) ter *SS* (»Slave Select«). Na vodilu je eni enoti, navadno procesorju, dodeljen poseben status »master«, ta enota krmili promet po vodilu. Žica za referenčno ničlo tudi tu navadno ni potrebna, saj so integrirana vezja na nivoju istega tiskanega vezja že priključena na skupno zemljo. Na sliki 4.11 je narisana bločna shema povezave SPI.



Slika 4.11: Povezava med enotami na vodilu SPI

Enota »master« (A) daje več »slave select« *SSn* signalov, za vsako na vodilo priključeno »slave« (B, C, D) enoto enega. S signalom *SSn* enota »master« aktivira tisto od na vodilo priključenih enot, s katero želi izmenjati podatke. Ko je izbrana enota aktivna, sledi pošiljanje podatkov po liniji *MOSI* od enote »master« proti izbrani enoti »slave« in po liniji *MISO* od izbrane enote »slave« proti enoti »master«. Prenos podatkov je sinhron s sunki na liniji *SCLK*: vrednost linij MOSI in MISO se sme spremeniti ob rastočem robu signala SCK, obe povezani enoti pa prebereta signal z linij ob padajočem robu signala SCK. Na sliki 4.12 zgoraj je narisan potek signalov pri prenosu podatkov od enote »master« do enote »slave«, na isti sliki spodaj pa pri prenašanju v obratni smeri. Obstaja alternativna verzija protokola, kjer je signal ure negiran, prav tako so zamenjani tudi trenutki spreminjanja linij MOSI in MISO ter vpisovanja v enoti.



Slika 4.12: Signali na vodilu SPI. Od naslovljene enote se do procesorja po liniji MISO prenaša vrednost 69_H, od procesorja do naslovljene enote pa po liniji MOSI vrednost $C6_H$.

Nivoji napetosti za prenos ničel in enic niso strogo definirani in so odvisni od uporabljenih integriranih vezij ter napajalnih napetosti. Prav tako ni strogo definirana hitrost prenosa podatkov, saj je odvisna od sposobnosti uporabljenih integriranih vezij. Ker izbrano enoto aktiviramo preko linij *SS*, naslova izbrane enote ni treba pošiljati po vodilu, zato je prenos lahko podatkov hitrejši. Za razliko od vodila I^2C pri vodilu SPI nagovorjena enota »slave« ne potrdi svoje prisotnosti na vodilu ali pravilnega prejema podatkov, zato je prenos manj zanesljiv.

4.2.4. Vodilo USB

Vodilo USB (»Universal Serial Bus«) je namenjeno pošiljanju podatkov med računalnikom in drugim, specializiranim sestavnim delom računalniškega sistema (tiskalnik, modem, spominska enota ter tudi vmesnik, ki je opremljen z lastnim procesorjem). Vodilo sestavljata dve liniji za podatke in dve liniji za napajanje, nič voltov in +5 voltov. Dve podatkovni liniji imata vedno komplementarno vrednost in zato omogočata hiter prenos podatkov. Promet na vodilu nadzira glavna (»master«) enota oziroma računalnik.

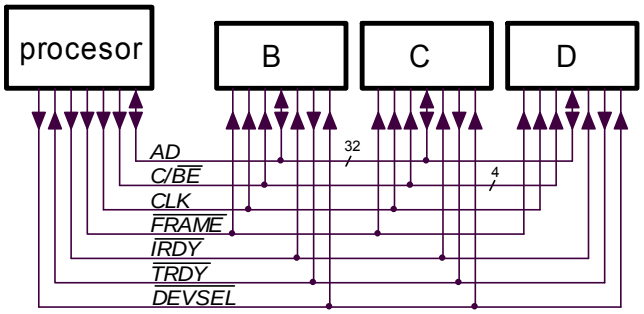
Način delovanja USB vodila je visoko standardiziran in ga nadzira specializiran procesor v povezanih enotah. Opis podrobnosti USB vodila presega namen te knjige. Povejmo le, da je za pošiljanje podatkov preko USB vodila potreben poseben program v obeh enotah, ki sta na vodilo priključeni. Mi ga bomo uporabljali le v povezavi s knjižnicami in gonilniki, kjer so ustrezni programi že zapisani.

Poudariti velja, da je USB vodilo namenjeno hitremu prenosu večje količine podatkov. To se doseže s koncentriranjem podatkov v priključenih enotah. Po vodilu se prenaša koncentrirane podatke v obliki paketov največ enkrat na milisekundo, kar utegne nagajati pri pogostem prenašanju majhnih količin podatkov med enotami.

4.3. Nekatera posebna vodila

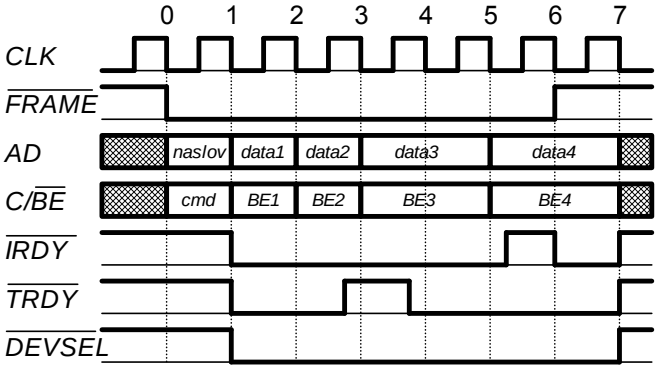
4.3.1. PCI (»Peripheral Component Interconnect«)

Se uporablja v osebnih računalnikih za povezavo med procesorjem in vmesniškimi (grafika, disk, zajemanje, USB,…) enotami. V osnovi je to vzporedno vodilo, kjer imamo opravka z 32 bitnim podatkovno-naslovnim vodilom *AD* in nizom dodatnih linij (*CLK*, *FRAME*, *C/BE*, *IRDY*, *TRDY* in *DEVSEL*), ki dopolnjujejo trenutne signale na podatkovno-naslovnem vodilu, slika 4.13. Pri prenosu podatkov po vodilu lahko izkoristimo polno širino vodila, lahko pa tudi samo poljuben bajt. Prenos podatkov je definiran v standardu in je sestavljen iz posameznih tipov prenosov, vsega 12. Nekateri tipi prenosa so namenjeni komunikaciji s spominskimi enotami, drugi konfiguraciji enot, nekateri prenašanju podatkov do vhodno-izhodnih enot in podobno. Podrobna razlaga je v $$$. Vsak tip prenosa se začne s posredovanjem naslova na vodilu *AD* in nadaljuje s prenosom podatkov po istem vodilu po sliki 4.14, kjer je za zgled prikazan niz signalov za branje. Prenos podatkov se lahko ponavlja. Na sliki 4.15 je prikazan prenos podatkov po istem vodilu za pisanje.



Slika 4.13: Povezava enot na vodilu PCI

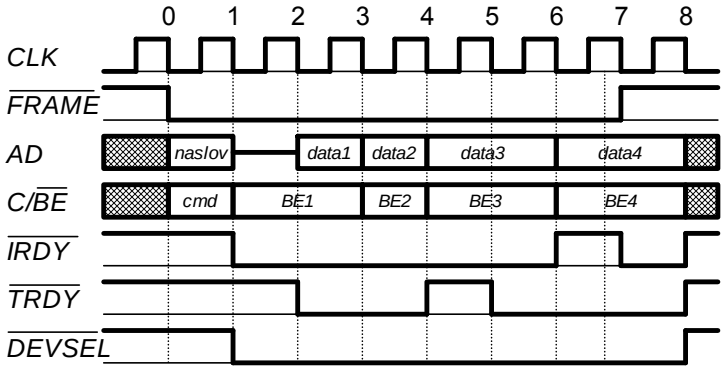
Prenos podatkov pri pisanju se začne takrat, ko nadzorna enota (morda procesor preko ustreznega logičnega vezja) na vodilo *AD* pošlje naslov enote in hkrati kodiran tip prenosa podatkov na linije *C/BE*. To stori sinhrono z uro *CLK*, začetek prenosa pa potrdi s signalom *FRAME*, ki dobi vrednost nič. Naslovljena enota potrdi svojo pripravljenost tako, da signal *TRDY* (»Target Ready«) spremeni na vrednost nič, hkrati tudi nadzorna enota potrdi svojo pripravljenost tako, da signal *IRDY* (»Initiator Ready«) spremeni na vrednost nič. Prenos se nadaljuje tako, da nadzorna enota pošlje na vodilo *AD* podatke za en cikel ure. Prenos se lahko ponavlja vsak cikel ure dokler je signal *FRAME* enak nič, če sta le obe enoti sposobni vzdržati tempo. Če katera od enot tempa ne zmore, to označi tako, da spremeni vrednost signala pripravljenosti (*IRDY* ali *TRDY*) na ena. Prenos se podaljša dokler nista spet obe enot pripravljeni, vendar ne več kot za v naprej definiran čas. Če prenosa podatkov ni mogoče opraviti, vodilo PCI javi napako. Prenos podatkov se konča, ko nadzorna enota spremeni vrednost signala *FRAME* na ena.



Slika 4.14: Pisanje iz procesorja v izbrano enoto na vodilu PCI

Prenos podatkov pri branju se začne podobno. Omeniti velja le dodaten čas enak eni periodi ure *CLK*, ki je potreben zaradi spremembe smeri na vodilu *AD*. V prvem ciklu namreč nadzorna enota pošilja na vodilo *AD* naslov, kasneje naslovljena enota

pošilja na vodilo *AD* podatke. Ta sprememba smeri zahteva en cikel ure *CLK*. Tudi tokrat enoti svojo pripravljenost za izmenjavo podatkov potrdita s signaloma *IRDY* in *TRDY*, kadar nista pripravljeni pa ta dva signala dobita vrednost ena. Prenos se mora opraviti v v naprej definiranem času, sicer vodilo PCI javi napako. Branje se konča, ko nadzorna enota vrne signal *FRAME* na ena.



Slika 4.15: Branje iz enote v procesor na vodilu PCI

Prekinitve so dopuščene preko posebnega tipa prenosa podatkov z imenom *INTA* in so preko dodatnih logičnih enot povezane s prekinitvenimi vhodi procesorja v osebnem računalniku.

Prenos podatkov po vodilu je sinhron s signalom ure *CLK*, ki znaša 33MHz, kar v kombinaciji s slikama 3.14 in 3.15 definira čas, potreben za prenašanje podatkov. Pri tem je ponavljanje prenašanja podatkov zelo učinkovito, saj je v tem primeru treba naslov prenesti samo enkrat. Teoretično je mogoče po vodilu PCI prenesti največ $33\ \text{MHz} \cdot 4\ \text{bajte na enkrat} = 132\ \text{Mbajtov v sekundi}$. Praktično je hitrost prenosa manjša, saj je treba prenašati še naslove, pa tudi priključene enote se morda odzivajo prepočasi in jih mora vodilo čakati.

Vodilo PCI je avtomatizirano do te mere, da se mora vsaka enota predstaviti vodilu. Predstavitev se opravi ob vklopu računalnika in vsebuje sporočanje osnovnih značilnosti enote (proizvajalec, standard, širina vodila, …).

Za enote, ki so vtaknjene v vodilo PCI, proizvajalci navadno napišejo niz rutin, ki omogočajo komunikacijo z njimi. Te rutine so podane kot gonilnik, ki ga instaliramo ob ali pred prvim zagonom računalnika z novo enoto. Pri programiranju enote se potem sklicujemo na rutine iz gonilnika, navadno jih kličemo z dodatnim setom rutin, ki je shranjen v priloženi knjižnici.

4.3.2. VME

Se uporablja v nekaterih visokozmogljivih fizikalnih merilnih sistemih. Vodilo je bilo razvito za uporabo v miniračunalnikih, vendar današnjim potrebam v zmogljivih računalnikih zaradi premajhnega pretoka podatkov ne zadošča več. Ker je bilo v

preteklosti razvitih veliko merilnih modulov in zanje ni potreben tak pretok podatkov, se vodilo VME še vedno veliko uporablja.

Vodilo VME sestavlja 32 bitno naslovno vodilo in 32 bitno podatkovno vodilo. Pretok podatkov po vodilu nadzira niz devetih kontrolnih signalov, največja hitrost prenosa podatkov pa je okrog 40 Mbajtov v sekundi. V večjih sistemih je širina vodil podvojena.

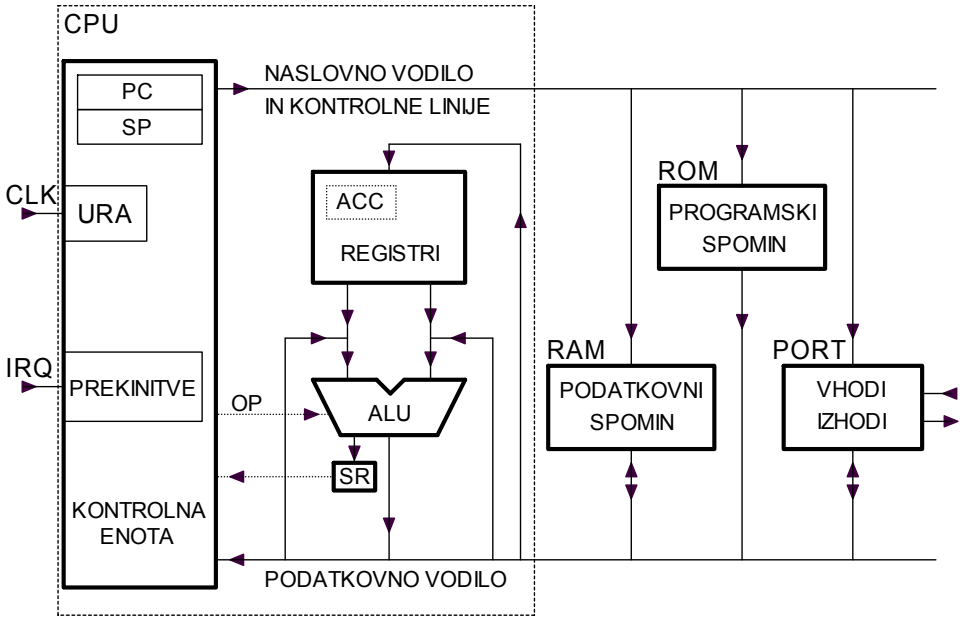
Vodilo je zasnovano tako, da lahko katerakoli enota prevzame nadzor. Posebna enota (»bus arbiter«, razsodnik vodila) dodeljuje vodilo tisti enoti, ki vodilo zahteva in je hkrati na vrsti za prevzem. Ko je enoti vodilo dodeljeno, le-ta poganja obe vodili ter kontrolne signale *RD* ter *WR*, ki definirajo smer prenosa podatkov. Naslovljena enota potrdi prejem podatkov tako s signalom na ustrezni kontrolni liniji.

Fizikalne merilne module navadno vstavljamo v ohišja, kjer je že prisotno vodilo in napajalnik. Pri vstavljanju enote moramo biti pozorni na naslov, ki ga taka enota zaseda v naslovnem prostoru; dve enoti ne moreta zasedati istega dela naslovnega prostora. V ta namen VME enote vedno vsebujejo elemente (stikala ali jahače), ki uporabniku omogočajo nastavitev naslova, na katerega se bo enota odzivala. Naslov je sestavljen in osmih šestnajstiških cifer, navadno pa nastavljamo le zgornji del naslova, saj enota zavzema blok v naslovnem prostoru.

Skupna poraba vseh enot, ki so vtaknjene v ohišje VME, ne sme presegati največjega dovoljenega toka, ki ga ohišje dopušča.

5. Procesor

Tipičen procesorski sistem sestavljajo vsaj bloki po sliki 5.1. Z vodili so prikazane povezave za pretok podatkov med bloki, puščice označujejo smer pretoka. Povezave, ki so potrebne za pravilno zaporedje opravljanja operacij, na sliki niso polno prikazane. V integriranem vezju – procesorju, so vsaj tisti moduli, ki so na sliki v črtkanem kvadratu CPU. Ostali moduli so prav tako nujni za delovanje procesorskega sistema, lahko so v integrirano vezje že vgrajeni ali pa so dodani izven integriranega vezja procesorja, z njim jih povezujeta naslovni in podatkovno vodilo ter kontrolne linije.

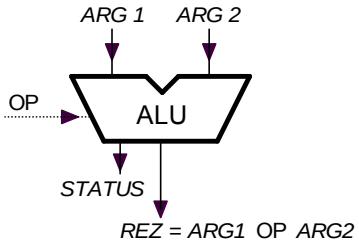


Slika 5.1: Zgradba tipičnega procesorskega sistema,
v procesorju so vsaj moduli v črtkanem pravokotniku - CPU

5.1. Centralna procesna enota - CPU

5.1.1. Aritmetična logična enota - ALU

Aritmetična logična enota je del procesorja, kjer se opravljajo vse matematične operacije. Sem sodijo na primer seštevanje, množenje, logične operacije AND, OR in podobno ter pomiki vsebine v levo ali desno. Simbol za ALU je na sliki 5.2. ALU za računanje potrebuje en ali dva argumenta *ARG*, ki sta pred izvajanjem shranjena v začasnih spominskih celicah – registrih ali pa prihajata preko podatkovnega vodila iz katerega od spominov. Rezultat *REZ* računanja se shrani nazaj v enega od registrov ali v spomin.



Slika 5.2: Simbol za ALU

ALU naenkrat opravlja eno od množice možnih operacij, izbere jo vrednost vodila OP. Vrednost vodila OP definira kontrolna enota na podlagi programa, po katerem deluje procesor in v njem zapisanega zaporedja operacij.

ALU enote različnih procesorjev se med sabo razlikujejo v naboru operacij, ki so jih sposobne izvajati in v številu bitov, ki jih lahko hkrati obdelujejo. Tipične so osem- in šestnajst- bitne ALU enote. Hitrost izvajanja operacij v ALU je prav tako različna in odvisna od tehnologije izdelave, velikosti nabora matematičnih operacij ter energije, ki je na razpolago za računanje. Hitrejši procesorji porabijo več električne energije, procesorji z več operacijami pa prav tako.

ALU je tipično sposobna izvajati več kot 30 različnih operacij, za katere lahko argumente dobi iz različnih virov. Moderni procesorji temeljijo na tako imenovanih RISC (»Reduced Instruction Set Computer«) tehniki, kjer se skuša optimirati delovanje procesorja in njegovo hitrost tako, da se zmanjša število operacij, kar omogoči hitrejše izvajanje le-teh.

5.1.2. Registri

Registri procesorja so osnovne spominske enote in so namenjeni začasnemu shranjevanju vrednosti spremenljivk. Število bitov v posameznem registru je prilagojeno ALU in velikosti naslovnega prostora procesorja. Če je v procesorju več registrov, je argumente računanja treba manj pogosto prinašati iz glavnega pomnilnika, vendar je zaradi tega zgradba procesorja bolj kompleksna. Tipični procesorji imajo od štiri do 32 registrov.

Nekateri registri so prilagojeni posebnim nalogam:

- Vsak procesor vsebuje register imenovan PC (»Program Counter«, programski števec). Vsebina tega registra je kazalec v programski spomin na naslov, kjer je shranjen bajt strojne kode, ki ga procesor trenutno izvaja. Ko procesor izvaja zaporedje strojnih ukazov (vrstice programa), se vsebina registra PC

sproti povečuje. Ko v programu pride do razvejitve, se vsebina PC prilagodi naslovu naslednjega strojnega ukaza. Register PC ima enako bitov, kot je žic v naslovnem vodilu (izjema so nekateri procesorji za osebne računalnike, kjer se vsebina naslovnega vodila formira s kombiniranjem dveh registrov).

- Vsak procesor vsebuje register z imenom SR (»Status Register«, register stanja). Ta je sestavljen iz niza bitov, ki jih imenujemo zastavica (»flag«). Zastavice označujejo trenutno stanje procesorja ali rezultat operacije v ALU. Sem sodijo na primer:
  - Prekoračitev obsega (»overflow«), zastavica označuje, da je rezultat zadnje matematične operacije presegel vrednost, ki jo je mogoče zapisati v register.
  - Ničla (»zero«), zastavica označuje, da je rezultat zadnje matematične operacije enak nič.
  - Prenos (»carry«), zastavica označuje, da je pri zadnji matematični operaciji prišlo do prenosa, ki ga je treba upoštevati pri naslednji matematični operaciji.
  - Negativna vrednost (»Negative«), zastavica označuje, da je rezultat zadnje matematične operacije negativnen.
  - Omogočanje prekinitev (GIE, »General Interrupt Enable«), zastavica omogoči prekinitve, njeno vrednost spreminja program, ki teče na procesorju.
- Vsak procesor vsebuje register z imenom SP (»Stack Pointer«, kazalec sklada). V podatkovnem spominu je za začasno shranjevanje vrednosti registrov rezervirano polje (sklad), vanj kaže vsebina registra SP. Ko v polje shranimo (zbirniški ukaz »Push«) vrednost kateregakoli registra, se vrednost kazalca sklada poveča za ena, ko iz polja preberemo (zbirniški ukaz »Pop«) vrednost, se vrednost kazalca sklada zmanjša za ena. Sklad in njegova vsebina sta zelo pomembna pri programiranju v zbirniku, pri višjih programskih jezikih je delovanje sklada pod nadzorom prevedenega programa in programerju nevidno.
- V nekaterih procesorjih je en od registrov imenovan Akumulator (»Accumulator«) in je bolje ožičen ter zato sodeluje v vseh operacijah ALU. Pri drugih procesorjih so registri enakovredni in katerikoli od njih lahko nastopa v operacijah.

5.1.3. Kontrolna enota

Kontrolna enota nadzira zaporedje izvajanja operacij po v naprej napisanem programu uporabnika in aktivira v procesor vgrajene enote v programu prilagojenem zaporedju. Kontrolna enota na podlagi rezultatov matematičnih operacij in vrednosti zastavic prinaša iz programskega spomina strojne kode za naslednje ukaze ter nadzira povečevanje vrednosti programskega števca.

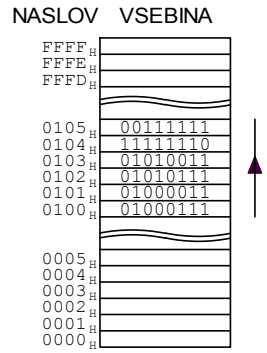
Strojna koda je niz ničel in enk, ki pomeni eno operacijo v procesorju. Pri tem strojna koda definira tudi argumente operacije, bodisi kot ime vpletenih registrov ali

direktno vrednost argumenta. V kontrolni enoti je vgrajen modul za dekodiranje, ki na podlagi strojne kode krmili kontrolne signale v procesorju tako, da se zahtevana operacija izvede.

Pri merjenju pogosto zahtevamo, da se določeni deli programa izvedejo takrat, ko električni signali dobijo določeno vrednost. Modul za prekinitve poskrbi, da se izvajanje običajnega programa prekine in začne izvajati poseben prekinitveni program takrat, ko signali IRQ dobijo izbrano vrednost.

Modul ure skrbi za pravilno časovno sosledje krmilnih signalov in definira hitrost izvajanja operacij v procesorju.

5.1.4. Programski spomin – ROM (včasih tudi »FLASH ROM«)



Slika 5.3: Spominski prostor shematsko in strojna koda, puščica označuje zaporedje izvajanja ukazov strojne kode

Strojna koda, torej prevedeni program za delovanje procesorja, je shranjena v programskem spominu. Strojna koda niz bajtov, ki vsebujejo ničle in enke. Vsaka kombinacija vrednosti bitov je ukaz za izvajanje drugačne operacije. Posameznemu bajtu ukaza lahko sledi bajt argumenta, ki je potreben za izvajanje operacije. To je lahko na primer naslov v podatkovnem spominu ali naslov registra kjer je argument shranjen ali pa kar argument sam.

Na sliki 5.3 je shematično narisan spominski prostor za program, ki se začne na naslovu 100_H. Program je v obliki bajtov zapisan na naslovih 100_H, 101_H, 102_H, … Če želimo ta program izvajati, najprej postavimo vrednost registra PC na 100_H, nato pa poženemo izvajanje. Kontrolna enota iz programskega spomina prebere strojni ukaz 47_H in ga izvede, nato prebere strojni ukaz iz lokacije 101_H, ki znaša 43_H in ga izvede in tako naprej. Če strojni ukaz potrebuje argument, potem kontrolna enota prebere najprej strojno kodo, nato pa še argument in šele nato začne z izvajanjem tega ukaza.

5.1.5. Podatkovni spomin - RAM

Podatkovni spomin služi shranjevanju večjih količin podatkov ali rezultatov, ki presegajo zmogljivosti registrov. Do podatkovnega spomina dostopamo preko obeh vodil in kontrolnih signalov, zato traja vpisovanje in branje dlje kot za registre. Podatkovni spomin obsega vsaj nekaj kB spominskih celic.

5.1.6. Vhodi in izhodi

Nekateri procesorski sistemi so opremljeni z moduli za vhode in izhode, ki so lahko digitalni ali analogni. Do teh modulov dostopamo preko obeh vodil in kontrolnih signalov tako, kot je bilo to opisano v poglavju o paralelnem vodilu.

5.2. Operacije v centralni procesni enoti

Za zgled napišimo program, ki se začenja na lokaciji 100H v programskem spominu, kot je to narisano na sliki 5.3. S programom v zanki povečujemo vrednost registra. Program bi lahko napisali v zbirniku (kaj pa je zdaj to?) in morda zgleda takole:

```
          ORG      100H          ; tu se začne program
          MOV.B    #0,R7         ; register R7 dobi vrednost 0
PONOVI    INC.B    R7            ;
          JMP      PONOVI
```

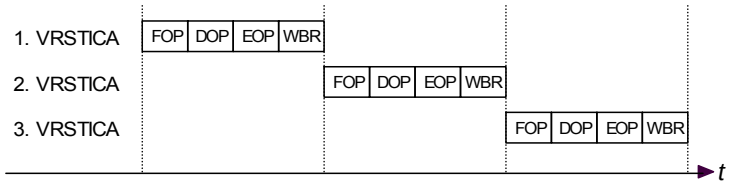
Na omenjeni sliki so ustrezno kodirani ukazi že vpisani v programski spomin. Vsaki vrstici programa ustrezata dva zaporedna bajta v programskem spominu. Procesorski sistem izvaja program po korakih, v vsakem je aktiven le del sistema, programski števec ima na začetku vrednost 100_H (zgled je osnovan na mikroprocesorju družine MSP430, ki ga uporabljamo pri praktičnem delu pouka). Koraki si sledijo v naslednjem vrstnem redu:

1. Dostava ukaza ali v angleščini »fetch opcode«, FOP: programski števec PC pošlje na naslovno vodilo vrednost 100_H, kontrolna enota pa zahteva branje iz ROM. Pride do branja kode iz ROM z izbranega naslova, vrednost 47_H gre v dekodirni modul, ki je del kontrolne enote. Ker kontrolna enota ve, da vsaki vrstici programa ustrezata (vsaj) dva bajta, poveča vrednost PC in postopek ponovi za naslov 101_H ter spet poveča vrednost PC tako, da njegova vsebina že kaže na ukaz, ki se bo izvajal kasneje.
2. Dekodiranje ukaza ali v angleščini »decode opcode«, DOP: v dekodirnem modulu kontrolne enote sta oba bajta, ki predstavljata vrstico programa. V tem koraku se dekodirna enota odloča za postopek, po katerem naj ukaz izvede. V procesu odločanja dekodira imena registrov, kjer so shranjeni argumenti za operacijo in pripravi vrednost za vodilo OP do enote ALU.
3. Izvajanje ukaza ali v angleščini »execute opcode«, EOP: kontrolna enota krmili delovanje registrov in ALU tako, da se izvede zahtevana operacija. Krmiljenje predstavlja niz signalov, ki najprej omogočijo prosto pot vsebini izbranih registrov ali podatkovnega spomina do ALU ter krmiljenje vodila OP, zaradi česar ALU izvede pravo operacijo. Ko je rezultat izračunan, kontrolna enota onemogoči pošiljanje argumentov v ALU.
4. Shranjevanje rezultata ali v angleščini »writeback result«, WBR: kontrolna enota omogoči izhod ALU tako, da se prej izračunani rezultat pojavi na podatkovnem vodilu ter krmili kontrolne signale in naslovno vodilo tako, da se ta rezultat vpiše v izbrani register ali podatkovni spomin.

Ko so opravljeni vsi štirje koraki izvajanja programske vrstice, procesor postopek ponovi od začetka, tokrat za vrednosti PC 102_H in 103_H ter tako naprej. Naslednji par bajtov (strojna koda), ki predstavlja naslednjo programsko vrstico zbirnika, je prenesena iz programskega spomina šele potem, ko so opravljeni vsi štirje koraki za tekočo vrstico.

Posebej velja omeniti le izvajanje zadnje od programskih vrstic, ki predstavlja preskok v zaporedju. V programski vrstici 'JMP PONOVI' se vrednost PC ne poveča, ampak zamenja z vrednostjo naslova v programskem spominu, kjer je shranjen ukaz 'INC.B R7'.

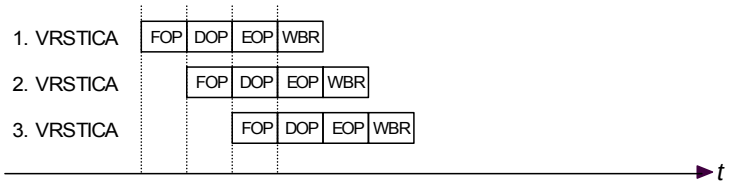
Tako izvajanje programskih vrstic se zdi manj racionalno. V posameznem koraku je aktiven le del procesorja, ostale enote pa čakajo, kar je shematsko prikazano na sliki 5.4. Izvajanje programa bi bilo hitrejše, če bi hkrati delale vse enote. To bomo poskusili v naslednjem poglavju.



Slika 5.4: Izvajanje programskih vrstic po korakih v procesorju

5.2.1. Optimizacija na strojnem nivoju

Operacije, ki jih mora izvesti kontrolna enota, lahko torej razdelimo na štiri korake: branje strojnega ukaza iz programskega spomina »FOP«, vrednotenje prebranega strojnega ukaza »DOP«, nadziranje ALU in registrov za izvršitev v strojni kodi zapisane operacije »EOP« ter shranjevanje rezultata »WBR«. Zato tudi kontrolno enoto razdelimo na štiri dele, ki izvajajo posamezno od zgoraj naštetih nalog. Ker gre za štiri ločene dele kontrolne enote, lahko vsi štirje deli delajo sinhrono: kadar četrti del nadzira shranjevanje rezultata, lahko tretji del že opravlja naslednjo operacijo. Hkrati lahko drugi del za dekodiranje strojne kode razmišlja o potrebnem postopku za izvajanje ene operacije kasneje in prvi del bere iz programskega spomina novo strojno kodo. Tako vzporedno branje, vrednotenje izvajanje in shranjevanje je shematsko prikazano na sliki 5.5. Govorimo o »pipeline« tehniki. Stopnja pohitritve je odvisna od tega, koliko in katere korake lahko procesor izvaja vzporedno.



Slika 5.5: Vzporedno izvajanje korakov pospeši izvajanje programa

Seveda ni nujno, da procesor izvaja program iz zaporednih lokacij v programskem spominu, saj so možne razvejitve, ki so posledica vrednosti rezultata ali programa samega. Torej šele ob koncu izvajanja operacije procesor ve, ali je naslednji ovrednoteni ukaz, ki čaka v tretji enoti res tisti, ki ga je treba v naslednjem koraku

izvesti. Če je pravi, ga kontrolna enota pošlje v izvajanje. Če ni pravi, se zavrže vsa vsebina kontrolne enote, register PC se postavi na novo lokacijo v programskem spominu in od tam se nadaljuje z izvajanjem. Nekateri procesorji imajo posebne enote, ki v naprej skušajo uganiti naslednji korak v izvajanju tudi takrat, ko bo morda prišlo do razvejitve. Govorimo o »prefetch« tehniki, ki zmanjša delež zavrženih ukazov v kontrolni enoti.

5.2.2. Nabor ukazov na strojnem nivoju

Pri praktičnem delu pouka bomo uporabljali procesor iz družine MSP430. V tabeli so navedeni vsi ukazi, ki jih procesor izvaja. Argumenti ukazov so lahko direktna števila, vsebina registra, vsebina podatkovnega spomina ali pa kazalci na vsebino podatkovnega spomina, ki so shranjeni v registrih.

ADD	Seštej dva argumenta,
ADDC	Seštej dva argumenta in upoštevaj prenos prejšnjega seštevanja
AND	Logična operacija AND
BIC	Spremeni bit v nič, »bit clear«
BIS	Spremeni bit v ena, »bit set«
BIT	Testiraj bit, »bit test«
CALL	Klic podprograma
CMP	Primerjaj argumenta
DADD	Seštej decimalno
JC	Skoči, če je rezultat prejšnje operacije povzročil prekoračitev obsega
JEQ	Skoči, če je rezultat prejšnje operacije nič
JGE	Skoči, če je rezultat zadnje primerjave večji ali enak nič
JL	Skoči, če je rezultat zadnje primerjave manjši od nič
JMP	Skoči v vsakem primeru
JN	Skoči, če je rezultat zadnje primerjave negativen
JNC	Skoči, če pri zadnji operaciji ni prišlo do prenosa
JNE	Skoči, če je rezultat zadnje primerjave od nič različen
MOV	Kopiraj vrednost argumenta
PUSH	Spravi bajt na sklad
RETI	Končaj prekinitveni podprogram
RRA	pomakni vsebino registra za eno mesto v desno
RRC	Rotiraj vsebino registra
SUB	Odštej
SUBC	Odštej in upoštevaj prenos prejšnjega odštevanja
SWPB	Zamenjaj bajta v besedi

SXT	Spremeni osembitno število v šestnajstbitno
XOR	Logična funkcija XOR

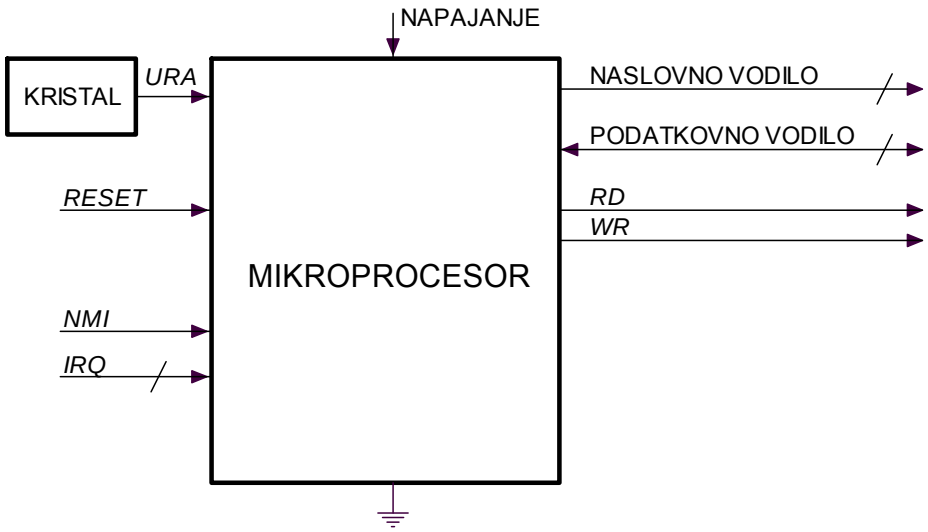
5.3. Mikroprocesor

Mikroprocesor je integrirano vezje, ki vsebuje vse navedene module iz s črtkano črto označenega dela slike 5.1 ter dodatne enote, ki omogočajo pravilno delovanje. To so na primer logična vezja za inicializacijo ob vklopu, vezja za nadzor napajalnih napetosti in podobno. Vse navedene enote so med sabo interno povezane s podatkovnim in naslovnim vodilom ter kontrolnimi linijami. Vsaka enota ime vgrajeno logiko za naslovno dekodiranje, širina podatkovnega in naslovnega vodila pa je prilagojena številu bitov, ki jih procesor hkrati obdeluje.

Naslovno in podatkovno vodilo sta izvedena na priključkih mikroprocesorja, prav tako sta tam tudi kontrolna signala *RD* in *WR*. Na obe vodili priključujemo enote tako, kot je bilo to opisano v poglavju o paralelnem vodilu.

Mikroprocesorji imajo navadno še priključke za napajanje ter priključke za druge funkcije. Med njimi tu omenimo le vhode za ponovni zagon (»RESET«), prekinitvene signale *NMI* (»Non Maskable Interrupt«, stalni prekinitveni vhod) in *IRQ* (»Interrupt ReQuest«, prekinitveni vhod), teh zadnjih je lahko več.

Nekateri mikroprocesorji že vsebujejo podatkovni in/ali programski spomin, večini pa ga priključimo na obe vodili in kontrolna signala.



Slika 5.6: Tipični priključki mikroprocesorja

5.3.1. DMA (»Direct Memory Access«)

Med enotami v mikroprocesorskem sistemu je včasih potrebno prenašati velike količine podatkov. Za primer: enota za zajemanje podatkov je samostojno pomerila vrednost vhodnega signala v enakomernih časovnih intervalih in sedaj vsebuje 1Mbajt podatkov. Te podatke je potrebno pred obdelavo kopirati v podatkovni spomin procesorskega sistema. To lahko storimo tako, da podatke enega za drugim preberemo iz enote za zajemanje v procesor in vpišemo iz procesorja v podatkovni spomin, kar je običajen postopek za pretakanje podatkov po vodilu. Taka pot je zamudna, saj je sestavljena iz dveh operacij (branja iz enote za zajemanje, nato pisanja v podatkovni spomin) ter sprotnega branja programa za procesor (ki ga spet sestavlja vsaj nekaj branj ukazov iz programskega spomina) in izvajanja istega programa.

Direktnega kopiranja podatkov med enotami na vodilu mimo procesorja običajen procesorski sistem ne dopušča. Potrebujemo posebno enoto DMA (»Direct Memory Access«), ki prevzame nadzor nad naslovnim in podatkovnim vodilom ter kontrolnima linijama *RD* in *WR*, potem je mogoče podatke med enotami kopirati brez vmešavanja procesorja.

Vzemimo, da želimo merilne podatke kopirati iz enote za zajemanje na naslovu 1000_H v podatkovni spomin na naslovu 8000_H, podatkov pa je za 2000_H bajtov. Enoto DMA zato najprej programiramo z naslovom vira podatkov (1000_H), naslovom, kamor naj se podatki prenesejo (8000_H) in količino podatkov za prenos (2000_H). Ko je to narejeno, sprožimo DMA prenos podatkov in s tem procesor odklopimo z vseh vodil in kontrolnih linij, ki jih prevzame enota DMA. Enota DMA vsebuje strojno opremo, ki samostojno bere in piše ter avtomatsko povečuje vrednost obeh kazalcev dokler niso vso podatki kopirani. Po kopiranju se enota DMA odklopi z vseh vodil in kontrolnih linij, nadzor pa ponovno prevzame procesor.

Ker je kopiranje opravljeno brez branja programskega spomina in vrednotenja strojne kode ter izvajanja programa v procesorju, je bistveno hitrejše.

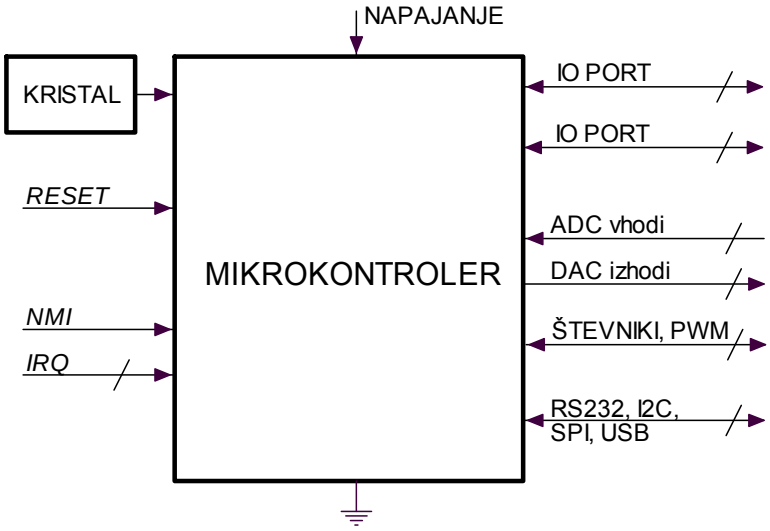
5.4. Mikrokontroler

5.4.1. Dodatni moduli v mikrokontrolerju

Mikrokontroler je prilagojena verzija mikroprocesorja, ki navadno že vsebuje določeno količino programskega in podatkovnega spomina. Interno je opremljen z vodili, ki te enote povezujejo, prav tako je že vgrajeno naslovno dekodiranje ter generator signala ure. Mikrokontroler je opremljen s priključki za povezavo v merilni sistem ter vsebuje dodatne module za merjenja, ki jih navadno v mikroprocesorju ne najdemo in so navedeni v nadaljevanju. Kompleten mikrokontroler kupimo v obliki integriranega vezja. Pogosto je to samostojna enota, ki ji dodamo le še napajanje in senzorje, napišemo program in že lahko začnemo meriti.

Mikrokontroler navadno ni tako hiter pri računanju, kot je mikroprocesor. Bolj je prilagojen merjenju, pri tem velika hitrost izvajanja operacij navadno ni potrebna.

Nabor tipičnih priključkov na mikrokontrolerju je na sliki 5.6.

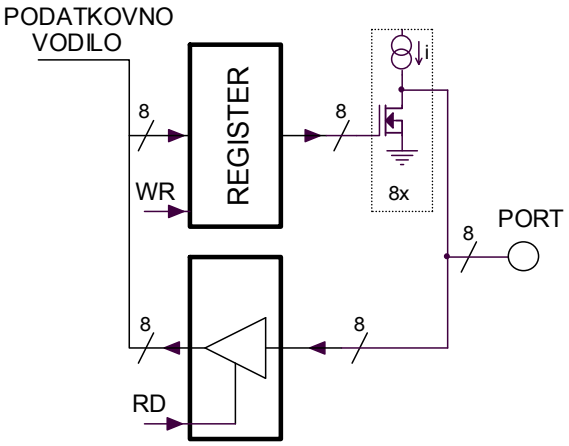


Slika 5.7: Tipični priključki mikrokontrolerja

Priključna vodila – port

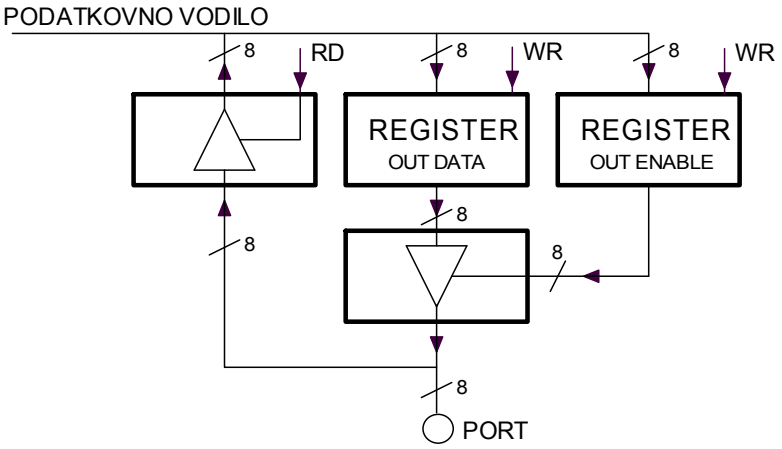
Kadar želimo na vodilo mikroprocesorja posredovati digitalni signal, potrebujemo enoto s tristanjskim izhodom in dekoder naslova po sliki #. Podobno potrebujemo register in naslovni dekoder tudi takrat, ko želimo iz podatkovnega vodila procesorja merilnemu sistemu posredovati digitalne vrednosti, ista slika. V mikrokontrolerju so ta vezja že vgrajena in na priključnih sponkah mikrokontrolerja so izvedeni izhodni signali registra in/ali vhodni priključki tristanjskega vmesnika s slike #. Niz osmih takih priključkov imenujemo priključno vodilo ali port. Zaradi optimizacije števila priključnih sponk mikrokontrolerja imajo navadno porti dvosmerno vlogo, s programskimi ukazi pa določimo smer prenosa podatkov. Pri tem sta v osnovi možni dve rešitvi.

- Pri nekaterih mikrokontrolerjih (družina Intel 8031) je port notranje povezan po sliki 5.8. Tu lahko procesor preko vodila in registra le potegne izhodno napetost na priključku PORT na logično nič tako, da aktivira tranzistor. Kadar tranzistor ne prevaja, je izhodna napetost zaradi šibkega tokovnega generatorja I enaka logični ena. Ker je tokovni generator šibek (<1mA), vezje na primer ni sposobno prižgati LE diode, ki je vezana od izhoda proti zemlji. Nasprotno tako vezje brez težav prižge LE diodo, ki je vezana proti napajanju. Kadar tranzistor ni aktiven, lahko signal PORT potegne na zemljo tja priključeno vezje, kar preberemo preko tristanjskega vmesnika (trikotnik) na podatkovno vodilo procesorja.



Slika 5.8:Interna povezava enote PORT, ki dopušča, da je isti priključek mikrokontrolerja uporabljen za vhod ali izhod, ne da bi bilo treba programsko spremeniti smer prenosa podatkov

- Pri nekaterih mikrokontrolerjih (Texas Instruments MSP430 ter Microchip PIC) je port povezan po sliki 5.9. Tu z vsebino dodatnega registra »OUT ENABLE« dočamo smer pretoka podatkov za vsak posamezen bit porta. Kadar je na primer bit 1 v registru »OUT ENABLE« enak logični ena, je aktiviran bit 1 tristanjskega registra, zato je ustrezni priključek rezerviran za izhodni signal iz mikrokontrolerja, vsi ostali priključki pa so rezervirani za vhodne signale. Pri taki razporeditvi le izhodni tok na nožici mikroprocesorja večji. Nekateri procesorji port obogatijo z dodatnimi funkcijami, zato je registrov še več.

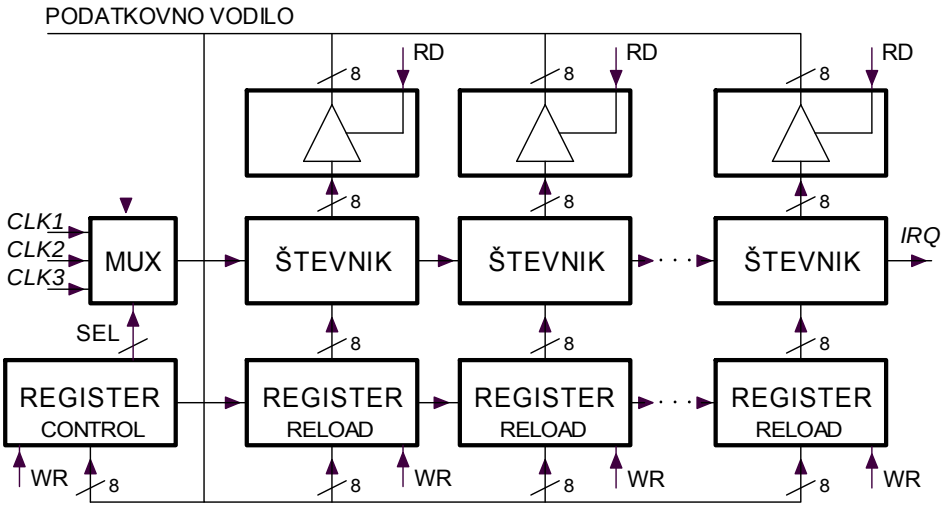


Slika 5.9: Interna povezava enote PORT, ki dopušča programsko izbiro smeri pretoka podatkov preko priključka mikrokontrolerja

Števnik

V merilnih sistemih se pogosto pojavi potreba po štetju sunkov. V ta namen procesorju dodamo števnik (oziroma niz osembitnih števnikov) in ga povežemo na podatkovno vodilo po sliki 5.10. Za to bi potrebujemo niz tristanjskih vmesnikov in naslovni dekoder, ki generira signale RD. Dodatni registri »RELOAD« omogočajo postavljanje števnika na izhodiščno vrednost, register »CONTROL« pa nadzira delovanje števnika. Multiplekser »MUX« omogoča štetje različnih signalov.

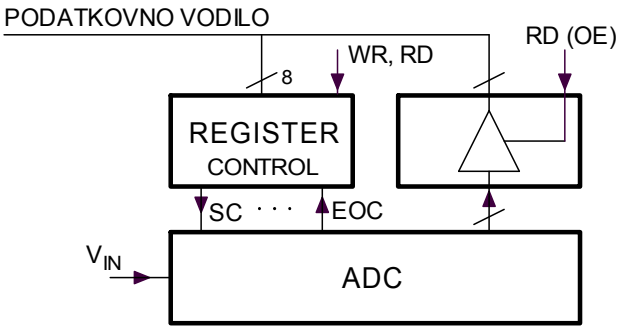
V mikrokontroler je navadno že vgrajen števnik, ki ima še mnoge druge funkcije. Podrobnosti si bralec lahko ogleda v podatkih za mikrokontroler, na primer MSP430. Tu naj navedemo le, da števnik lahko šteje sunke, ki jih priključimo na izbrano nožico mikroprocesorja, šteje lahko sunke notranje ure mikroprocesorja in tako definira časovne intervale, povzroča lahko prekinitve delovanja mikroprocesorja, generira lahko pulzno-širinsko moduliran signal *PWM* in še mnogo drugih. Pogosto je v mikrokontrolerju več takih števnikov.



Slika 5.10: Števnik v mikrokontrolerju in priključitev na podatkovno vodilo

ADC

Tudi analogno digitalni pretvornik lahko dodamo procesorskemu sistemu tako, da ga preko tristanjskega vmesnika povežemo na podatkovno vodilo po sliki 5.11. Dodati moramo še naslovno dekodiranje za branje rezultata pretvorbe in naslovno dekodiranje ter register za nadzor delovanja pretvornika. Vse to je v mikrokontrolerju že vgrajeno, na zunanje sponke mikrokontrolerja je treba le še priključiti merjeni signal in napisati program za zajemanje podatkov.

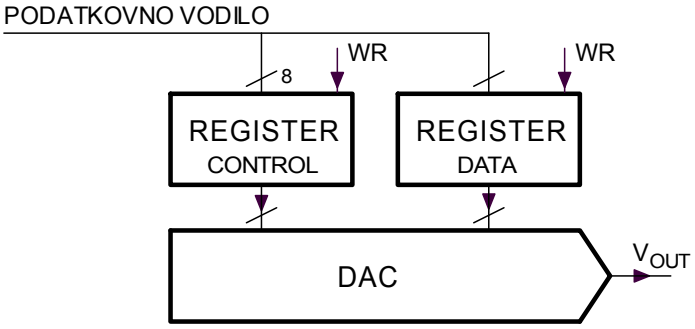


Slika 5.11: Priključitev ADC na vodilo procesorja

DAC

Če želimo, da procesor generira analogni signal, ga moramo opremiti z digitalno analognim pretvornikom. Tega lahko povežemo na podatkovno vodilo procesorja po sliki 5.12, kjer je narisano še vezje za naslovno dekodiranje in tristanjski vmesniki za dostop do vodila ter naslovno dekodiranje in register za nadzor delovanja digitalno analognega pretvornika.

V mikrokontroler je lahko tako vezje že vgrajeno, na nožici mikrokontrolerja je dostopna le še izhodna napetost digitalno analognega pretvornika. Seveda je treba napisati program za nadzor delovanja pretvornika.

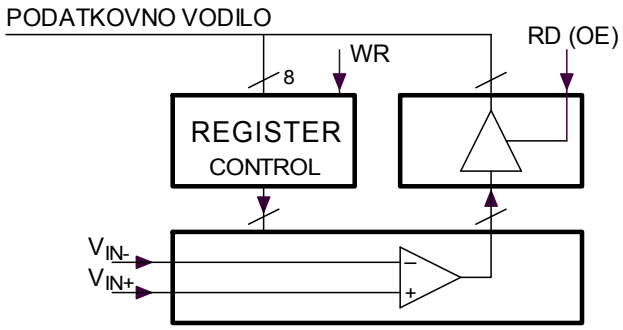


Slika 5.12: Priključitev DACa na vodilo procesorja

Komparator

Komparator je verzija analogno digitalnega pretvornika, ločevati zna le med dvema diskretnima nivojema. Kljub temu je v merinih sistemih lahko zelo koristen. Ima en izhodni in dva vhodna priključka. Če želimo komparatorjev izhodni signal prebrati s procesorjem, ga lahko povežemo na podatkovno vodilo po sliki 5.13, potrebujemo tristanjski vmesnik ter vezje za naslovno dekodiranje, s katerim

aktiviramo tristanjski vmesnik (RD ali OE) od branju s pravega naslova. Dodatni register služi za nastavljanje parametrov delovanja komparatorja.



Slika 5.13: Tako priključimo izhod komparatorja na vodilo procesorja

Vezja za serijsko komunikacijo

Mikrokontroler pogosto navezujemo na inteligentne senzorje, ki že vsebujejo vezja za analogno obdelavo signala in analogno digitalni pretvornik. Zdi se smiselno, da tak inteligentni senzor opremimo z vezjem, ki omogoča prenašanje podatkov do oddaljenega procesorja. Mikrokontroler lahko tudi navežemo na osebni računalnik tako, da mikrokontroler služi za zbiranje in vrednotenje informacij, osebni računalnik pa za interakcijo z uporabnikom. V ta namen je v mikrokontroler že vgrajeno vezje, ki tako serijsko povezavo omogoča. Vezje vsebuje niz registrov, njihova vsebina določa način in parametre delovanja vezja za serijsko komunikacijo.

Za primer: v mikrokontrolerju MSP430F1611 sta na razpolago dva modula za serijsko komunikacijo, ki dopuščata prenos podatkov po standardih RS232, I2C in SPI, vsi potrebni priključki za komunikacijo pa so izvedeni na nožicah mikroprocesorja. Potrebnega je le nekaj programiranja. V drugih mikrokontrolerjih najdemo tudi module, ki omogočajo USB komunikacijo.

6. Programski jeziki

Procesor izvaja strojno kodo, ki je sestavljena iz nizov ničel in enic. Strojna koda je shranjena v programski memoriji računalnika in procesor jo od tam prinaša bajt za bajtom, kolikor pač potrebuje za sprotno delo. Strojna koda je človeku težko razumljiva, zato smo uvedli njene ekvivalente na enostavnem in bolj abstraktnem nivoju, to so programski jeziki.

6.1. Zbirnik (»assembler«)

Je programski jezik, ki je direktna preslikava strojne kode procesorja in je prilagojen izbranemu procesorju. Če izberemo drug procesor, bo verjetno treba uporabiti zbirnik z drugačnimi ukazi. V tem zapisu bomo uporabljali zbirnik za družino mikroprocesorjev MSP430 firme TI, s tem mikroprocesorjem se bomo ukvarjali tudi pri praktičnem delu pouka.

Enemu ukazu strojne kode ustreza en ukaz v zbirniku. Ukaze v zbirniku zapišemo s črkami, ki so okrajšava za željeno operacijo, tem dodamo še argumente operacije. Tako na primer zbirniški ukaz MOV.B R6,#10 pomeni, da naj procesor v svoj register z oznako R6 vpiše (MOV, premakni oziroma zapiši) vrednost 10_{10}. Pri navedenem zbirniku za procesor MSP430 dodatek '.B' pomeni, da je argument operacije bajt, dodatek '.W' pa za argument izbira besedo. Popoln nabor in opis ukazov zbirnika bralec najde v $$$.

V zbirniku napisan program je lažje berljiv od strojne kode in dobremu programerju omogoča optimalno izrabo procesorjevih zmogljivosti. Zato ključne dele kompleksnih programov vedno pišemo v zbirniku. Žal je zbirnik kompliciran, zato pri običajnih programih raje uporabljamo višje programske jezike.

Postopek prevajanja iz zbirnika v strojno kodo za izbrani procesor je avtomatiziran in ga opravljajo posebni programi – prevajalniki, ki lahko tečejo tudi na osebnih računalnikih.

Zgled programa v zbirniku za MSP430:

6.2. Višji programski jeziki

Višji programski jeziki so bolj prilagojeni razmišljanju človeka in omogočajo kontrolo izvajanja strojnih operacij v mikroprocesorju na višjem nivoju. Prav tako dopuščajo imenovanje spremenljivk ter v ozadju poskrbijo za interakcijo s strojno opremo in ostalimi programi, ki morda tečejo na procesorju. Programiranje v višjih programskih jezikih je enostavnejše, a nikoli optimalno hitro ali minimalno po obsegu potrebnega programskega spomina.

6.2.1. C

Je univerzalen jezik za programiranje tako osebnih računalnikov kot mikroprocesorkih sistemov in mikrokontrolerjev. Ukazi se s pomočjo prevajalnika prenesejo v strojno kodo, ki jo razume mikroprocesor. Ker je proces prevajanja prilagojen uporabljanemu mikroprocesorju, isti program v C-ju lahko implementiramo na različnih mikroprocesorjih. Paziti pa je treba na podrobnosti, kot so na primer:

- Količina spomina in lokacija v naslovnem prostoru za podatke, program in sklad, to je navadno definirano v datotekah, ki so priložene prevajalniku. Preveri datoteko 'MSP430F16x.xcf' za nastavitve pri prevajalniki IAR.
- Dolžina spremenljivk tipov char in int je na različnik mikroprocesorjih lahko različna! Večina prevajalnikov za osebni računalnik je spremenljivka tipa char 16 bitna, spremenljivka tipa int pa 32 bitna. Na mikrokontrolerju MSP430 sta tipa osem oziroma 16 bitna.
- Vsako sklicevanje na strojno opremo mikrokontrolerja je treba prilagoditi uporabljenemu mikrokontrolerju (prekinitve, ADC, DAC, števniki, …), sintaksa pa je prilagojena prevajalniku.
- Hitrost izvajanja programa je odvisna od sposobnosti procesorja in pri kritično hitrih delih programa je treba to upoštevati.

6.2.2. Visual Basic – VB6

Je namenjen programiranju osebnih računalnikov. Je dobro grafično podprt in omogoča enostavno konstruiranje uporabniškega vmesnika. Zanj so na razpolago knjižnice za dostop do v osebni računalnik vključenih enot ter strojne opreme, zato je programiranje prenosa podatkov in nadzora teh enot enostavno.

Program v programskem jeziku Visual Basic sestavljajo objekti in akcije. Objekti so večinoma vidni uporabniku in sestavljajo uporabniški vmesnik. Objektom pripišemo lastnosti bodisi v času pisanja programa ali kasneje. Akcije od dogodkih v programu (intervencija uporabnika, čas, komunikacija) popišemo z besedilom, ki uporablja standardne ukaze za prireditve, matematične funkcije in razvejitve.

Podatki so lahko shranjeni v različnih oblikah; kot celoštevilčne vrednosti, realne vrednosti ali konstrukti.

6.2.3. LabView

7. Programiranje – triki

7.1. Bitne logične operacije

Bitne logične operacije se v izvajajo na posameznem bajtu ali med istoležnimi biti dveh bajtov. Če je argument operacije en sam in je shranjen v registru, ga rezultat nadomesti. Če sta argumenta bitne logične operacije dva in podana v dveh registrih, rezultat operacije nadomesti vsebino enega od registrov z argumenti. V višjih programskih jezikih rezultat bitne logične operacije shranimo v poljubno spremenljivko, ki pa mora biti pravega tipa.

7.1.1. Bitna inverzija

Bitno inverzijo uporabljamo takrat, ko želimo negirati vrednost vseh bitov v bajtu. Argument v spodnjem zgledu je $5A_H$.

		0	1	0	1	1	0	1	0
NOT									
		1	0	1	0	0	1	0	1

Zgled v zbirniku MSP430:

```
MOV.B   R6,#H5A  ; vpiši 01010101₂ v register R6
NOT.B   R6       ; negiraj R6, rezultat A5H gre v R6
```

Zgled v Cju:

```
char a = 0x5A;   // deklaracija in inicializacija
a = ~a;          // negiraj a, rezultat A5H gre v a
```

Zgled v VB6:

```
Dim a As Byte    ' deklaracija argumenta
a = &H5A         ' inicializacija: a = 5AH
a = Not(a)       ' negiraj a, rezultat A5H gre v a
```

7.1.2. Bitni OR

Bitno operacijo OR med drugim uporabljamo takrat, kadar moramo spremeniti vrednost enega ali več bitov v bajtu na ena, ostale bite pa ohraniti. Argumenta v spodnjih zgledih sta števili $8E_H$ in 10_H.

		1	0	0	0	1	1	1	0
OR		0	0	0	1	0	0	0	0
		1	0	0	1	1	1	1	0

Zgled v zbirniku MSP430:

```
MOV.B   R7,#10001110b   ; vpiši 10001110₂ v register R7
MOV.B   R6,#00010000b   ; vpiši 00010000₂ v register R6
OR.B    R7,R6           ; OR,   10011110₂ gre v R7
```

Zgled v Cju:

```
char a = 0x8E;          // deklaracija in inicializacija
char mask = 0x10;       // deklaracija in inicializacija
a = a | mask;           // a = 9E_H
```

Bitne operacije OR z operatorjem '|' v programskem jeziku 'c' ne smemo zamenjati z logično operacijo OR, ki je označena z operatorjem '||'.

Zgled v VB6:

```
Dim a As Byte           ' deklaracija argumenta a
Dim mask as Byte        ' deklaracija argumenta mask
a = &H8E                ' inicializacija: a = 8E_H
b = &H10                ' inicializacija: b = 10_H
a = a OR mask           ' bitni OR, rezultat gre v a
```

7.1.3. Bitni AND

Bitno operacijo AND med drugim uporabljamo takrat, ko želimo izolirati vrednost izbranega bita iz bajta ali pa enega ali več bitov v bajtu spremeniti na nič, ostale bite pa ohraniti. Argumenta sta tokrat $8E_H$ in 09_H.

	1	0	0	0	1	1	1	0
AND	0	0	0	0	1	0	0	1
	0	0	0	0	1	0	0	0

Zgled v zbirniku MSP430:

```
MOV.B   R7,#10001110b   ; vpiši 10001110₂ v register R7
MOV.B   R6,#00001001b   ; vpiši 00001001₂ v register R6
AND.B   R7,R6           ; AND,  00001000₂ gre v R7
```

Zgled v Cju:

```
char a = 0x8E;          // deklaracija in inicializacija
char mask = 0x09;       // deklaracija in inicializacija
a = a | mask;           // a = 08_H
```

Zgled v VB6:

```
Dim a As Byte           ' deklaracija argumenta a
Dim mask as Byte        ' deklaracija argumenta mask
a = &H8E                ' inicializacija: a = 8E_H
mask = &H09             ' inicializacija: mask = 09_H
```

```
   a = a And mask           ' bitni AND, rezultat gre v a
```

Bitne operacije AND z operatorjem '&' v programskem jeziku C ne smemo zamenjati z logično operacijo AND, ki je označena z operatorjem '&&'.

7.1.4. Postavljanje in podiranje istega bita v bajtu

Večkrat se pojavi potreba, da isti bit v bajtu najprej spremenimo na ena, kasneje pa spremenimo spet na nič. Definiranju dvojnih bitnih mask za spreminjanje vrednosti na ena in nič se ognemo, če uporabimo logične operacije po spodnjem zgledu.

Zgled v zbirniku MSP430:

```
   MOV.B   R7,#10001110b   ; vpiši 10001110₂ v register R7
   MOV.B   R6,#00010000b   ; vpiši 00010000₂ v register R6
   OR.B    R7,R6           ; OR, v R7 postavi bit 4
   CPL.B   R6              ; negiraj R6
   AND.B   R7,R6           ; AND, v R7 podri bit 4
```

Zgled v Cju:

```
   char a = 0x8E;          // a = 8E_H
   char mask = 0x10;       // mask = 10_H
   a = a | mask;           // spremeni bit 4 na ena
   a = a & ~mask;          // spremeni bit 4 na nič
```

Zgled v VB6:

```
   Dim a As Byte           ' deklaracija argumenta a
   Dim mask as Byte        ' deklaracija argumenta mask
   a = &H8E                ' inicializacija: a = 8E_H
   mask = &H10             ' inicializacija: mask = 10_H
   a = a Or mask           ' bitni OR spremeni bit 4 na ena
   a = a And Not(mask)     ' bitni AND spremeni bit 4 na nič
```

Z vrednostjo spremenljivke 'mask' izberemo bit, ki ga želimo spreminjati, z operacijo OR ta bit spremenimo na ena, z operacijo AND in negiranjem vrednosti spremenljivke 'mask' pa bit vrnemo na vrednost nič.

7.2. Razvejitve programa zaradi vrednosti bita

Procesor navadno izvaja programske ukaze enega za drugim, tako kot je to predvidel in zapisal programer. Možne so le razvejitve, kjer se procesor na podlagi programerjevega zapisa in vrednosti spremenljivk odloči za eno od možnih nadaljevanj izvajanja programa. Za nas je pomembno, da lahko potek programa temelji na vrednosti bita v bajtu. Tudi tu je najenostavneje uporabiti bitne logične funkcije za izolacijo vrednosti bita. Spodnji zgledi kažejo, kako se to programira v različnih programskih jezikih.

Zgled v zbirniku MSP430: če ima bit 3 registra R7 vrednost ena nadaljuj z izvajanjem programa na oznaki DA, sicer nadaljuj na oznaki NE.

```
   BIT.B   R7,#00001000b   ; testiraj bit 3 registra R7
   JZ NE                   ; nadaljuj…
DA:                        ; tule, če je bit vreden ena
```

```
   -
   -
NE:                        ; tule, če je bit vreden nič
   -
   -
```

Zgled v Cju:

```
   char a = 0x8E;                // a = 8E_H
   char mask = 0x08;             // mask = 10_H
   if (a & mask == mask)  {      // če je bit 3 argumenta a ena
      -                          // postori tole
      -
   }
   else {
      -                          // sicer postori tole
      -
   };
```

Zgled v VB6:

```
   Dim a As Byte              ' deklaracija argumenta a
   Dim mask as Byte           ' deklaracija argumenta mask
   a = &H8E                   ' inicializacija: a = 8E_H
   mask = &H10                ' inicializacija: mask = 10_H
   If (a And mask) Then       ' če je bit 3 argumenta a ena
      -                       ' postori tole
      -
   Else
      -                       ' sicer postori tole
      -
   Endif
```

Tudi zanke lahko ponavljamo, dokler bit v bajtu ne dobi ustrezne vrednosti. V zgledu čakamo, da bit 2 dobi vrednost ena, potem izstopimo iz zanke.

Zgled v zbirniku MSP430: čakamo, da dobi bit 2 v registru R12 vrednost ena.

```
   MOV.B   R12,#0          ; inicializiraj R12
L:
   BIT.B   R12,#00000100b  ; testiraj bit 2 registra R12
   ADD.B   R12,#1          ; dodaj registru ena
   JZ L                    ; ostani v zanki, če…
```

Zgled v Cju:

```
   char a = 0x00;                // a = 00_H
   char mask = 0x04;             // mask = 04_H
   do { a++; }
   while ((a & mask) != mask);
```

Zgled v VB6:

```
   Dim a As Byte              ' deklaracija argumenta a
   Dim mask as Byte           ' deklaracija argumenta mask
   a = 0                      ' inicializacija: a = 0
   mask = &H4                 ' inicializacija: mask = 4_H
```

```
Do                          ' čakaj v tej zanki
     a = a + 1
Loop While (a And mask)        ' dokler se pogoj ne izpolni
```

7.3. Kazalec v polje

Pri shranjevanju ali obdelavi niza izmerkov imamo pogosto opravka s polji, kamor so izmerki shranjeni. Za elemente polja je v pomnilniku računalnika rezerviran prostor, na izbrani element polja pa kaže kazalec. Število elementov v polju je omejeno, zato moramo pri rabi polj biti previdni. Če poskušamo pisati v polje, pa kazalec po pomoti kaže ven iz prostora, ki je za polje rezerviran v spominu računalnika, bomo popisali del spomina, ki polju ne pripada. Podobno lahko preberemo vrednost, ki ne pripada polju. Oboje lahko katastrofalno vpliva na izvajanje programa, zato je ves čas treba paziti, da kazalec ostaja v za polje rezerviranem območju. To lahko storimo z dvema stavkoma IF; če je kazalec prevelik, ga zmanjšamo, če je premajhen, ga povečamo. Lažje in hitreje gre, če uporabimo logične funkcije.

pt	pt_2	pt & 11_2	$(pt \,\&\, 11_2)_{10}$
5	00000101_2	01_2	1
4	00000100_2	00_2	0
3	00000011_2	11_2	3
2	00000010_2	10_2	2
1	00000001_2	01_2	1
0	00000000_2	00_2	0
-1	11111111_2	11_2	3
-2	11111110_2	10_2	2
-3	11111101_2	01_2	1
-4	11111100_2	00_2	0
-5	11111011_2	11_2	3
-6	11111010_2	10_2	2

Tabela $$: kazalec omejimo s funkcijo AND

Vzemimo, da imamo opravka s poljem, ki ima 2^N elementov, pri tem je N celo število. Če je N na primer 2, ima kazalec na polje lahko le vrednosti 00_2, 01_2, 10_2 in 11_2, ostale niso dovoljene. Spremenljivka, ki predstavlja kazalec v polje, je navadno dolga en bajt ali eno besedo. Če od tega vzamemo za dejanski kazalec le zadnja dva bita, ta ne more kazati ven iz polja. Ko povečujemo vrednost kazalca, dejanski kazalec najprej kaže na element nič, nato na element ena, dva, tri in potem spet na element nič. Tudi v obratni smeri gre enako. Ko vrednost kazalca zmanjšujemo, kaže dejanski kazalec najprej na element nič, nato na element tri (in ne minus ena), potem na element dva, element ena in spet na element nič.

Ta trik deluje, kadar je polje sestavljeno iz 2^N elementov. V programu je zato smiselno izbrati tako dolžino polja, zgled VB6 je:

```
DIM arr(32) as Integer, pt as Integer
```

```
pt = 0
DO                                      ' ponavljaj
   arr(pt And &h1F) = Rnd * 256         ' vpiši naključno število
   pt = pt + 1                          ' povečaj kazalec
   pt = pt And &h7FFF                   ' vendar ne preko 32767
loop
```

Najprej deklariramo polje 'arr' z 32 elementi, njihovi indeksi so od 0 do 31, nato inicializiramo kazalec 'pt' na vrednost nič in v neskončni zanki povečujemo vrednost kazalca ter v elemente polja vpisujemo naključne vrednosti. Kazalec ke tu omejen na najmanj pomembnih pet bitov z ukazom 'And &h1F'. S predzadnjo vrstico poskrbimo, da vrednost kazalca ne preseže vrednosti 32767, kar je največja dovoljena pozitivna vrednost za spremenljivke tipa 'integer'.

7.3.1. Krožni pomnilnik

Navadno za merilne rezultate rezerviramo del procesorjevega pomnilnika, ki pa ni neskončno velik. Kadar zajemamo neprekinjeno, se zanašamo na to, da merilnih rezultatov čez nekaj časa ne bomo potrebovali in jih zato lahko popišemo z novimi rezultati, saj bomo iz starih v vmesnem času že izluščili značilnosti. Takrat govorimo o krožnem pomnilniku, ki je navadno polje, le kazalec v polje je omejen na način, ki je bil opisan v prejšnjem podpoglavju. Grafično je tak pomnilnik prikazan na sliki #, sestavlja pa ga 2^N elementov, kar olajšuje dostop do elementov in skupaj s povedanim v prejšnjem podpoglavju preprečuje pisanje ali branje izven za rezultate rezerviranega območja v podatkovnem spominu.

Izberemo seveda dovolj dolgo polje tako, da imamo na voljo dovolj časa za obravnavo starih zajetih rezultatov preden jih popišemo z novimi. Vzemimo, da moramo računati frekvenčni spekter zajetega signala, vzorci tega signala prihajajo v enakomernih časovnih intervalih, za računanje spektra potrebujemo 32 vzorcev. V tem primeru izberemo krožni pomnilnik, ki ima 64 elementov. Ta pomnilnik začnemo polniti pri elementu z indeksom nič. Ko zberemo 32 vzorcev in je torej prvih 32 elementov popisanih, sprožimo podprogram za računanje spektra teh 32 vzorcev ter sočasno nadaljujemo s shranjevanjem vzorcev v zgornjo polovico krožnega pomnilnika. Za računanje spektra imamo na razpolago toliko časa, kot ga potrebujemo za zajemanje 32 vzorcev, potem se vzorci spet shranjujejo v spodnjo polovico krožnega pomnilnika, spekter pa računamo iz vzorcev, zajetih v zgornji polovici pomnilnika.

7.4. Razvejitve programa in vrednosti signalov, priključenih na procesor

Procesorski sistem naj bi se odzival na signale iz okolice. Vzemimo, da mora procesor ob spremembi vrednosti signala (bit 0 spremenljivke RQ) postoriti v programu nekaj posebnega. Program za procesor lahko napišemo tako, da v zanki ves čas preverja vrednost spremenljivke RQ. Ta je lahko na primer bajt, ki ga preberemo z vhodnih priključkov procesorja. Ko izbrani bit spremenljivke RQ dobi vrednost ena,

program opravi potrebno nalogo in počaka, da se signal RQ vrne na prvotno vrednost, nato spet ponavlja preverjanje vrednosti signala RQ. Ustrezen program v jeziku 'C' je:

```
char RQ;
char RQmask = 0x01;
do {
   RQ = portA;                          // beri port procesorja
   If (RQ & RQmask == RQmask)           // če je treba…
      Opravi_nalogo ();                 // reagiraj in
While (1);                              // ponovi test
```

Tak način programiranja je preveč potraten, saj procesor pogosto preverja stanje spremenljivke RQ, kasneje pa čaka na prenehanje zahtevka RQ. Želeli bi, da procesor poleg preverjanja vrednosti spremenljivke RQ počne še kaj drugega, to omogočimo na dva načina.

7.4.1. Poizvedovanje (»Polling«)

Procesor naj preverja stanje spremenljivke le v bolj ali manj enakomernih časovnih intervalih. Prej navedena zanka v osnovi ostaja, dodan je le še del, v katerem procesor opravi ostale, za uporabnika koristne naloge (Ostale_naloge ();).

```
char RQ;
char RQmask = 0x01;
do {
   RQ = portA;                  // beri port procesorja
   If (RQ & RQmask == RQmask)   // če je treba…
      Opravi_nalogo ();         // reagiraj in
   Ostale_naloge ();            // opravi še ostale naloge in
While (1);                      // ponovi test
```

Te ostale naloge lahko razdelimo na manjše kose (Ostale_naloge0, Ostale_naloge1, Ostale_naloge2, Ostale_naloge3) tako, da ob vsakem obhodu skozi zanko procesor opravi en kos te naloge, pri tem za vsak kos naloge potrebuje približno enak čas. To enostavno rešimo z uvedbo spremenljivke za štetje k, kateri se ob vsakem obhodu skozi zanko vrednost poveča za ena in stavkom »switch«, ki na podlagi vrednosti te spremenljivke opravlja kose.

```
char RQ;
char RQmask = 0x01;
char k = 0;
do {
   RQ = portA;                          // beri port procesorja
   If (RQ & RQmask == RQmask)           // če je treba…
      Opravi_nalogo ();                 // reagiraj,
   Else                                 // sicer
      Trati_čas ();                     // potrati enako časa in
   Switch (k++)  {                      // opravi ostale naloge
   Case 0: Ostale_naloge0 ();  break;       // kos 0
   Case 1: Ostale_naloge1 ();  break;       // kos 1
   Case 2: Ostale_naloge2 ();  break;       // kos 2
```

```
    Case 3: Ostale_naloge3 ();  break;       // kos 3
    };
    k &= 0x03;                          // ter omeji števec in
While (1);                              // ponovi test
```

Približno enakomerno časovno izvajanje programa lahko zagotovimo, če dodamo še prazen podprogram Trati_čas, ki se izvaja enako dolgo kot podprogram Opravi_nalogo, vendar v resnici samo trati procesorjev čas. Spet se zdi smiselno, da nalogo razdelimo na 2^N kosov (N je celo število) zaradi enostavnejšega omejevanja vrednosti števca k.

7.4.2. Prekinitev (»Interrupt«)

Procesor ponuja tudi boljše poti za preusmerjanje izvajanja programa, imenujemo jih prekinitve. Gre za to, da izvajanje programa v mikroprocesorju zmotimo z zunanjim električnim signalom. Ta povzroči, da začne procesor izvajati posebej v ta namen pripravljeni prekinitveni podprogram. Ko je izvajanje prekinitvenega podprograma končano, procesor nadaljuje s programom, ki ga je izvajal pred prekinitvijo. To je simbolno prikazano na sliki #.

Prekinitve in s tem povezano programiranje je močno odvisno od procesorja, ki ga uporabljamo. V splošnem velja sledeče.

- Prekinitveni podprogram moramo v naprej napisati in ga shraniti v programskem spominu procesorja.
- Procesor mora vedeti, kje v spominu je prekinitveni podprogram, ker ga bo le tako lahko začel ob prekinitvi izvajati, nekateri procesorji zahtevajo prekinitvene podprograme na točno določenem mestu v spominu, drugi spet se zadovoljijo s kazalcem na prekinitveni podprogram.
- Procesorju moramo dovoliti (omogočiti) izvajanje prekinitvenega podprograma, nekaterim procesorjem lahko nastavimo še parametre izvajanja.
- Procesor mora nekako označiti, da je zahteva po izvajanju prekinitvenega podprograma uresničena, navadno to pomeni postavljanje ali podiranje zastavice. Do označevanja lahko pride avtomatsko od začetku ali koncu izvajanja prekinitvenega podprograma, v nekaterih primerih pa mora prekinitveni program vsebovati ukaze za označevanje. Šele po označenju mehanizem za prekinitve prisluhne novim zahtevam.

Časovni potek Izvajanje programa v procesorju ponazorimo z diagramom na sliki #. Navpično šrafirani del nad absciso predstavlja izvajanje osnovnega programa. Ko pride do prekinitve, ki jo označuje puščica pod absciso, procesor ne izvaja več osnovnega programa, ampak nadaljuje s prekinitvenim podprogramom, ki je ponazorjen s poševno šrafiranim področjem. Izvajanje prekinitvenega podprograma je bolj pomembno od izvajanja osnovnega programa, da ima torej višjo prioriteto. Po koncu izvajanja prekinitvenega podprograma se izvajanje vrne na osnovni nivo.

Če do nove prekinitve pride še v času izvajanje prekinitvenega podprograma, se zahteva shrani in povzroči ponovno izvajanje takoj, ko je to mogoče. Ta situacija je na narisana na isti sliki desno.

Procesor ima lahko več vhodnih priključkov za prekinitvene signale, nekatere prekinitve lahko povzročijo tudi moduli v notranjosti procesorja, na primer detektor napačne napajalne napetosti, števniki in komunikacijski moduli. Ker niso vsi prekinitveni signali različnih enot enako pomembni, jim dodelimo prioriteto izvajanja. Velja, da prekinitveni signal z višjo prioriteto lahko prekine izvajanje kateregakoli programa ali podprograma, ki ima nižjo prioriteto. Taka situacija je nakazana na sliki # levo, kjer izvajanje osnovnega programa najprej zmotimo s prekinitvijo nizke prioritete, nato pa izvajanje ustreznega prekinitvenega podprograma zmotimo s prekinitvijo visoke prioritete. Ko je podprogram za prekinitev visoke prioritete končan, procesor nadaljuje z izvajanjem podprograma za prekinitev nizke prioritete, nato se vrne na izvajanje osnovnega programa.

Če najprej pride do prekinitve visoke prioritete in nato med izvajanje prekinitvenega podprograma visoke prioritete do prekinitve nizke prioritete, se podprogram za prekinitev visoke prioritete izvede do konca, nato procesor nadaljuje z izvajanjem podprograma za prekinitev nizke prioritete in se na koncu posveti izvajanju osnovnega programa. Taka situacija je nakazana na sliki # desno.

Pri nekaterih procesorjih so prioritetni nivoji nastavljivi s programom, pri drugih so prekinitvenim vhodom dodeljene fiksne prioritete.

Zgled za implementacijo prekinitvenega podprograma v jeziku 'c' za mikroprocesor MSP430 firme TI. Za druge procesorje gre drugače!

```
// prekinitveni podprogram
#pragma vector=PORT1_VECTOR          // prekinitev na portu
__interrupt void Port_1 (void){      // začetek
   Stori_tole ();
   P1IFG = 0;                        // označi konec
}

// glavni program
void main (void) {
   P1IE = 0x01;               // omogoči prekinitve na portu
   _BIS_SR(GIE);              // omogoči prekinitve CPU
   for (;;)       {};         // neskončna zanka
}
```

7.5. Branje ali pisanje bita

Signali na podatkovnem vodilu procesorskega sistema se ves čas spreminjajo. Kadar želimo iz procesorskega sistema nadzorovati signal na žici, moramo na vodilo priključiti register in poskrbeti za naslovno dekodiranje, ki v register sproži vpis vrednost z vodila takrat, ko programska oprema zahteva vpis na izbrani naslov. Eden od izhodov tega registra je želeni signal.

Vzemimo, da je v naslovnem prostoru procesorskega sistema prost naslov 500_H in želimo iz tega sistema pisati bit, ki ga bomo v register pošiljali kot najmanj pomemben bit po podatkovnem vodilu. Sistemu dodamo register in naslovno dekodiranje po sliki #. Ustrezna programska oprema, ki spremeni vrednost signala najprej v ena in zatem v nič ter to ponavlja, je:

Zgled v zbirniku:

```
A:
   MOV.B   &500H,#1        ; spremeni signal v ena
   MOV.B   &500H,#0        ; spremeni signal v nič
   JMP   A                 ; ponavljamo
```

Zgled v C-ju:

```
   void main (void) {
   unsigned int Signal @ 0x500;  // pove, da je spremenljivka
                                 // Signal na naslovu 500H
      for (;;) {                 // ponavljamo
         Signal = 1;             // Signal gor
         Signal = 0;             // Signal dol
      };
   }
```

V obeh primerih ostane vrednost signala enaka ena le kratek čas, ki je odvisen od hitrosti izvajanja programa. Ker želimo spreminjati le vrednost enega signala, bi namesto registra lahko uporabili en sam flip-flop tipa D, naslovno dekodiranje pa ostane enako. V programu bi lahko vpisovali vse bite bajta hkrati, saj se ostali tako ali tako ne shranijo v en sam flip-flop.

Na isti sliki je narisano tudi vezje za branje posamičnega bita v procesorski sistem. Tokrat je dodan osem bitni tristanjski vmesnik, ki sliši na naslov 500_H, uporabljen pa je le en sam bit na najmanj pomembno linijo podatkovnega vodila. Kadar programska oprema zahteva branje z naslova 500_H, naslovni dekoder aktivira linijo MI, ki v povezavi s signalom RD omogoči tristanjski vmesnik in pošlje signal Not na podatkovno vodilo procesorskega sistema, od koder ta potuje v procesor na obdelavo.

Če želimo hkrati pisati ali brati več signalov, uporabimo polni register in tristanjski vmesnik, takrat s pomočjo programske opreme in prej podanih logičnih funkcij izoliramo iskane bite v prebranem bajtu.

7.6. Branje ADC

Kadar je analogno digitalni pretvornik (ADC) priključen na vodilo mikroprocesorja, moramo poznati njegov naslov in format, v katerem ADC vrača izmerjeno vrednost. Vzemimo, da imamo opravka s dvanajst bitnim pretvornikom in je strojna oprema procesorskega sistema sestavljena tako, da se ADC odziva na naslov 310_H (LSB) in 311_H (MSB). Biti rezultata so razporejeni po sliki #. ADC potrebuje še kontrolni signal SC, z njim določimo trenutek zajemanja (prehod iz nič v ena sproži postopek

zajemanja) ter daje signal EOC, s katerim DAC določi trenutek, ko je zajemanje končano (nič: zajemanje v teku). Vzemimo, da sta kontrolna signala ADCja navezana na vodilo procesorskega sistema po sliki #. Signal SC je bit 0 registra, ki ga vpisujemo z vodila in je na naslovu 312_H, signal EOC pa lahko preberemo preko tristanjskega vmesnika, ki sliši na naslov 312_H.

Za zajem moramo napisati program, ki je sestavljen iz treh delov:

a) Sproži zajemanje tako, da signal SC spremeniš iz ena v nič in nazaj v ena
b) V zanki preverjaj stanje EOC in čakaj na konec zajemanja
c) Ko dobi signal EOC vrednost ena, preberi rezultat v dveh korakih, vsaj bajt posebej
d) Sestavi bajta v skupni rezultat

Zgled v zbirniku:

```
Read_ADC:
   MOV.B   &312H,#0        ; SC na nič
   MOV.B   &312H,#1        ; SC na ena -> start
L:
   MOV.B   R10,&312H       ; beri EOC bit
   BIT.B   R10,#1          ; testiraj LSB registra R12: EOC bit
   JZ L                    ; ostani v zanki, če EOC == nič
   MOV.B   R10,&310H       ; beri LS bajt
   MOV.B   R11,&311H       ; beri MS bajt
   SWPB    R11             ; zamenjaj bajta v besedi R11, MS bajt
   ADD.W   R11,R10         ; prištej LS bajt, rezultat v R11
```

Zgled v C-ju:

```
   unsigned int Read_ADC (void)  {
   unsigned int adcLS      @ 0x310;     // definiraj naslov ADC
   unsigned int adcMS      @ 0x311;     // definiraj naslov ADC
   unsigned int adc        @ 0x312;     // definiraj naslov ADC
   unsigned int rezultat;
      adc = 0;                          // SC jo nič
      adc = 1;                          // SC je ena
      do {}                             // čakaj na EOC
      while (adc & 0x01 == 0x01);
      rezultat = adcLS + adcMS << 8;    // beri in sestavi
      return (rezultat);
   }
```

Kadar je ADC priključen na porte mikrokontrolerja po sliki $, je programiranje podobna, le naslovi za vpisovanje in branje so prilagojeni naslovom porta. Kadar imamo opravka s šestnajst bitnim procesorskim sistemom, oba bajta rezultata preberemo v enem koraku in njuno kombiniranje ni potrebno.,

7.7. Pisanje DAC

Vzemimo, da imamo opravka z dvanajst bitnim digitalno analognim pretvornikom (DAC), ki je priključen na osem bitno podatkovno vodilo tako, da na naslovu 300_H

prejema v svoj register spodnjih osem bitov podatka, na naslovu 301_H pa zgornje štiri bite podatka. Kadar imamo opravka z več kot osem bitnim pretvornikom, moramo torej vanj vpisovati dvakrat, vsaj bajt posebej. Da ne pride do napačnih izhodnih napetosti DACa, se ta dva bajta vpišeta v vmesni register, ki je del DACa, od tu pa ju z dodatnim ukazom prepišemo v pretvorniški del DACa in dobimo na izhodu ustrezajočo analogno napetost. Bločna shema vezja je na sliki #.

Pri programiranju najprej zapišemo spodnji bajt, nato zgornjega, za konec pa še podamo ukaz za prepis obeh bajtov iz vmesnega registra v pretvorniški del DACa. V zgledu vpisujemo v DAC na naslovu 340_H (LS) in 341_H (MS) bajt, ki se ob vsakem obhodu skozi zanko povečata. Z vpisom na naslov 342_H prenesemo vrednost iz registrov DACa v pretvorniški del. DAC zato generira žagasto napetost.

Zgled v zbirniku:

```
   MOV.W   R12,#0          ; inicializiraj R12
L:
   MOV.B   &H34,R12        ; LS bajt na naslov 34H
   SWAP    R12             ; zamenjaj bajta v R12
   MOV.B   &H35,R12        ; MS bajt na naslov 35H
   SWPB    R12             ; zamenjaj bajta v R12
   MOV.B   &342H,#0        ; prenesi v pretvorniški del
   INC.W   R12             ; povečaj vrednost
   JMP     L               ; in ponavljaj
```

Zgled v C-ju:

```
   MOV.BR12,#0             ; inicializiraj R12
L:
   BIT.BR12,#00000100b     ; testiraj bit 2 registra R12
   ADD.BR12,#1             ; dodaj registru ena
   JZ L                    ; ostani v zanki, če…
```

Kadar imamo opravka s šestnajst bitnim podatkovnim vodilom, je pisanje enostavnejše, saj podatek vpišemo v DAC v enem koraku. V tem primeru tudi ni potreben dodatni ukaz za prepis podatka iz vmesnega registra v pretvorniški del DACa; na njegovem izhodu se nova analogna vrednost pojavi takoj po vpisu.

Zgled v zbirniku:

```
   MOV.BR12,#0             ; inicializiraj R12
L:
   BIT.BR12,#00000100b     ; testiraj bit 2 registra R12
   ADD.BR12,#1             ; dodaj registru ena
   JZ L                    ; ostani v zanki, če…
```

Zgled v C-ju:

```
   MOV.BR12,#0             ; inicializiraj R12
L:
   BIT.BR12,#00000100b     ; testiraj bit 2 registra R12
   ADD.BR12,#1             ; dodaj registru ena
   JZ L                    ; ostani v zanki, če…
```

7.8. Operacija MAC – »multiply & accumulate«

Gre samo v zbirniku

Kot bomo videli v nadaljevanju tega zapisa je pri filtriranju signalov pogosto treba računati konvolucijo dveh signalov, torej je treba računati vsoto niza zmnožkov dveh števil. Program za računanje je lahko:

```
   MOV.B R12,#0            ; inicializiraj R12
L:
   BIT.B R12,#00000100b    ; testiraj bit 2 registra R12
   ADD.B R12,#1            ; dodaj registru ena
   JZ L                    ; ostani v zanki, če…
```

Pri filtriranju v realnem času je treba operacijo, ki se izvaja v zanki, opraviti čim hitreje. Žal hitrost izvajanja omejuje hitrost procesorja in hitrost branja strojnih ukazov iz programskega spomina. Če bi lahko vsebino zanke izvedli ne da bi vsak ukaz posebej prinašali iz programskega spomina, bi bilo izvajanje nedvomno hitrejše.

Tako izvajanje opisane zanke omogoča zbirniški ukaz MAC (»Multiply and accumulate«), ki v enem koraku opravi obe operaciji množenja in seštevanja. Program se zato poenostavi v:

```
   MOV.B R12,#0            ; inicializiraj R12
L:
   BIT.B R12,#00000100b    ; testiraj bit 2 registra R12
   ADD.B R12,#1            ; dodaj registru ena
   JZ L                    ; ostani v zanki, če…
```

Primerjava hitrosti izvajanja?

Tole boš moral natančneje pogledati za MSP430. Kako gre s kazalci?

8. Periodično zajemanje signalov

8.1. Pod kontrolo programske opreme

Pri periodičnem zajemanju vzorcev mora biti časovni interval med zaporednimi vzorci dobro definiran, sicer pride do napak, kakršne so bile opisane v poglavju $$. Če take napake ne motijo, lahko časovni interval med zaporednima izmerkoma definira programska oprema. Potrebujemo le prazno zanko, ki jo procesor izvaja med zaporednimi vzorčenji, število obhodov skozi prazno zanko pa med kalibracijo sistema prilagodimo zahtevanemu časovnemu intervalu med izmerki. Zgled za takšen program je podan spodaj. Vsakokrat zajamemo 16 vzorcev vhodnega signala, rezultat shranimo v polje R*ez*.

Zgled v zbirniku:

```
Define Rez   equ   1000H          ; tule bodo rezultati
Define Delay equ   80H            ; čas med izmerki

   …                              ; začetek
Take_16:                          ; ime podprograma za zajemanje
   MOV.B   R14,#16                ; toliko izmerkov rabimo
   MOV.W   R13,#Rez               ; kazalec na polje z rezultati
Se:
   CALL    Read_ADC               ; pomeri
   MOV.W   @R13,R11               ; shrani v polje
   INC.W   R13                    ; povečaj kazalec
   INC.W   R13                    ; za dva bajta
   MOV.B   R15,#Delay             ; preberi kasnitev
D1:
   DEC.B   R15                    ; telo zanke
   JNZ     D1                     ; že opravljeno?
   DEC.B   R14                    ; vsi izmerki v polju?
   JNZ     Se                     ; ne še, ponovi
   …                              ; da, nadaljuj drugje
```

Zgled v C-ju:

```
   unsigned int Rez[16];          // polje za rezultate
   const Delay = 0x80;            // tole določi med kalibracijo

   void Take_16 (void)     {
   char kaz, k;
```

```
   for (kaz = 0; kaz < 16; kaz++)     {      // za 16 izmerkov
      Rez[kaz] = Read_ADC ();                // pomeri & shrani
      for (k = 0; k < Delay; k++) {};        // kasnitev
   };
}
```

Časovni interval med izmerkoma je odvisen od hitrosti delovanja procesorskega sistema, ki je navadno konstantna, saj jo definira kristal kremena v generatorju urinih sunkov. Če je zgoraj napisani program edini program, ki ga procesor izvaja, je interval navadno dovolj dobro definiran. Žal procesorski sistem pogosto prisluhne prekinitvam, ki zmotijo delovanje osnovnega programa in na videz podaljšajo njegovo izvajanje. Ker v naprej ne vemo kako pogosto bodo prekinitve, ne moremo predvideti časa med dvema zaporednima izmerkoma. Zgornji program je zato uporaben le v omejenih primerih.

8.2. Pod kontrolo strojne opreme

Časovni interval med zaporednima izmerkoma mnogo bolje definira zunanji prožilni sunek, ki začne pretvorbo v ADC. Ko ADC opravi pretvorbo, procesor shrani rezultat pretvorbe v polje. Prednosti tega načina dela so:

- Časovni interval je pod nadzorom strojne opreme in je konstanten.
- Procesor je med dvema izmerkoma prost in lahko opravlja druge naloge.

Naslednji zgled kaže, kako programirati procesor MSP430, da pretvorbo sproži sunek na portu P1, najmanj pomemben bit.

```
unsigned int Rez[16];                  // polje za rezultate
char k = 0;                            // kazalec v polje

// prekinitveni podprogram
#pragma vector=PORT1_VECTOR            // prekinitev na portu
__interrupt void Port_1 (void){        // začetek
   Rez[kaz] = Read_ADC ();             // pomeri & shrani
   if (++kaz == 16)                    // je zajetih 16 vzorcev?
      _BIC_SR(GIE);      // da, zato onemogoči prekinitve
   P1IFG = 0;            // označi konec prekinitvene rutine
}

// glavni program
void main (void) {
   P1IE = 0x01;                // omogoči prekinitve na portu
   _BIS_SR(GIE);               // omogoči prekinitve CPU
   for (;;)       {};          // neskončna zanka
}
```

Zgornji zgled temelji na tem, da so sunki za proženje pretvorb, priključeni na port P1, periodični. Periodične sunke lahko pri mikrokontrolerjih dobimo tudi tako, da vklopimo števec, ki je v mikrokontroler vgrajen. Števec programiramo tako, da šteje do znane vrednosti, ko jo doseže, sproži prekinitev in s tem zajem vhodnega signala z ADC.

V zgledu sicer procesor samo teče v prazni zanki med čakanjem na novo prekinitev.

Procesor na prekinitev reagira hitro, vendar ne takoj. Pred reagiranjem mora dokončati operacijo (strojno kodo), ki jo ravnokar izvaja. Šele, ko je operacija opravljena, preskoči na izvajanje prekinitvenega podprograma. Posamične operacije ne trajajo enako dolgo, zato od prihoda sunka za prekinitev do dejanske prekinitve traja različno dolgo. Odzivni čas je odvisen tudi od tega, v kateri fazi izvajanja operacije pride prekinitveni sunek. Ti časi so kratki (morda do nekaj mikro sekund), vendar lahko motijo. Zato s sunkom raje direktno sprožimo pretvorbo ADCja in hkrati povzročimo prekinitev izvajanja programa procesorja. Ker je pretvorba sprožena direktno, je kasnitev minimizirana. Procesor potem samo prebere rezultat pretvorbe iz ADCja in shrani rezultat.

Zgled:

9. Serijska komunikacija med računalnikom in mikrokontrolerjem

Sistem za zajemanje in obdelavo podatkov je lahko zasnovan iz dveh delov. Osebni računalnik ponuja izdelano strojno opremo in programsko okolje, ki je primerno za interakcijo z uporabnikom in shranjevanje rezultatov ter hitro računanje. Žal zaradi kompleksnega operacijskega sistema manjka možnost programskega definiranja točnih časovnih intervalov, zato to nalogo navadno zaupamo posebej zgrajeni merilni enoti, ki temelji na mikroprocesorju. Ta merilni enota je bistveno enostavnejša od osebnega računalnika in njeni nalogi sta zajemanje podatkov in prenašanje le-teh v osebni računalnik, kjer jih nadalje obdelamo in skladiščimo. Bločna shema tako sestavljenega sistema je na sliki 9.1.

Slika 9.1: Bločna shema merilnega sistema z osebnim računalnikom in enoto za zajemanje

Kadar je merilni sistem sestavljen po zgornji sliki, moramo napisati programa za zajemanje na strani mikroprocesorja v merilni enoti in za interpretacijo podatkov, shranjevanje in interakcijo z uporabnikom za strani osebnega računalnika. Poleg tega moramo napisati še programa za obe enoti, ki omogočata pretok podatkov in ukazov med enotama. V tem poglavju se bomo osredotočili na pisanje programa za komunikacijo med enotama, uporabili pa bomo serijsko komunikacijo RS232, ki je dostopna na večini mikroprocesorjev. Na sodobnih osebnih računalnikih redko najdemo priključek za RS232, vendar je na tržišču mogoče dokupiti kabel, ki USB vodilo reducira na RS232, tak RS232 priključek pa s pomočjo gonilnikov, ki so del

operacijskega sistema osebnega računalnika, vidimo kot navaden RS232, slika 9.2 levo. Alternativno so mnogi merilni vmesniki opremljeni z integriranimi vezji, ki opravljajo pretvorbo iz USB v RS232 vodilo, slika 9.2 desno. Spet moramo na strani mikrokontrolerja in na strani osebnega računalnika programirati prenos podatkov po standardu RS232.

Slika 9.2: Tako kombiniramo vodilo USB na strani osebnega računalnika z vodilom RS232 na strani mikrokontrolerja.

Nekateri mikrokontrolerji so opremljeni z strojno opremo, ki omogoča komuniciranje po vodilu USB. Programiranje pretoka podatkov po vodilu USB je bistveno bolj kompleksno in presega namen tega zapisa.

Pred pisanjem moramo izbrati protokol prenosa podatkov. Poleg podatkov se med napravama namreč prenašajo še ukazi in podatki, ki jih morata obe enoti pravilno interpretirati. Napisati je torej treba program za komunikacijo za obe v prenos vključeni napravi.

Ukaze lahko kodiramo v posamične bajte. Če imamo opraviti z zajemanjem podatkov z osem bitnim ADCjem, lahko na primer črka 'M' pomeni ukaz osebnega računalnika, naj mikrokontroler zajame en izmerek, črka 'P' pa, naj zajeto vrednost pošlje v osebni računalnik. Program na strani mikrokontrolerja je lahko napisan tako, da vsak prejeti ukaz potrdi, na primer s črko 'D'. Tako bi na primer komunikacija med osebnim računalnikom in mikrokontrolerjem za zajem enega samega izmerka bila sestavljena iz štirih delov. V prvem delu osebni računalnik pošlje mikrokontrolerju črko 'M', ki jo mikrokontroler prejme in interpretira, zato odgovori osebnemu računalniku s črko 'D' in z ADCjem pomeri vhodni signal. Kasneje osebni računalnik pošlje črko 'P', ki jo mikrokontroler spet prejme in interpretira ter se odzove z vračanjem izmerjene vrednosti.

Argumente ukazov pošiljamo kot posamezni bajt ali niz bajtov, ki sledijo ukazu. Programa na obeh enotah morata biti napisana tako, da oddajni del opremi poslani ukaz s pravimi argumenti, sprejemni del pa prestreže in interpretira poslane argumente.

9.1. Programska oprema na strani osebnega računalnika

Za začetek preverimo, ali je osebni računalnik opremljen s strojno opremo za komunikacijo po RS232 protokolu in poiščemo ime, ki ustreza izbranemu komunikacijskemu kanalu, dosegljivem preko konektorja na ohišju računalnika. Oboje opravimo v operacijskem sistemu, ko zaženemo program »Device manager«. Pod opcijo »Ports« najdemo listo imen komunikacijskih kanalov, ki so na razpolago. Njihova imena se začno s »COM« in končajo z zaporedno številko, tipično »COM1«, »COM2« in dalje.

Delovanje komunikacijskega kanala preverimo s pomočjo terminalskega programa (Hyperterminal v WIN XP ali podobno) in osciloskopa. Ko na tastaturi pritiskamo tipke, terminalski program pošilja iz računalnika signale, ki ustrezajo na tastaturi pritisnjeni tipki. Z osciloskopom opazujemo signal na priključku TX tistega konektorja, ki ustreza izbranemu kanalu. Pred preverjanjem nastavimo komunikacijske parametre terminalskega programa. Sem sodijo ime izbranega kanala, hitrost prenosa podatkov, število bitov in podobno.

Ko pišemo program za komunikacijo na strani osebnega računalnika, uporabimo enega od višjenivojskih jezikov.

Odpri komunikacijski kanal

1. Pošlji znak, ki predstavlja ukaz
2. Počakaj na odziv
3. Ukrepaj
4. Zapri komunikacijski kanal

Zgled v VB6

9.2. Programska oprema na strani mikrokontrolerja

Preveri ali je komunikacijski kanal na razpolago

Inicializiraj strojno opremo

Čakaj na ukaz (polling ali prekinitev)

Odgovori / ukrepaj

Zgled v Cju za MSP430

10. Osebni računalnik in zvočna kartica

10.1. Zvočna kartica – vgrajena enota

Pri osebnem računalniku se zvočna kartica sama od sebe ponuja za zajemanje signalov, saj je vgrajena v vsak osebni računalnik. Ker so zvočne kartice prilagojene zajemanju zvočnih signalov, so žal pogosto za meritve drugih signalov preslabe. V tem poglavju bomo popisali lastnosti zvočnih kartic in njihovo uporabo.

10.1.1. Lastnosti

Zvočna kartica je namenjena zajemanju zvočnih signalov v frekvenčnem razponu od 20Hz do 20kHz in amplitudo do nekaj 100mV. Nekatere zvočne kartice slabše zajemajo signale s frekvencami blizu obeh zgoraj navedenih meja, kar je treba pred uporabo preveriti.

Frekvenca vzorčenja je tipično 44,1kHz, na razpolago pa sta dva kanala, preko katerih lahko vzorčimo. Pri tem velja, da je hitrost vzorčenja konstantna, programsko je mogoče izbirati le nekaj diskretnih vrednosti, ki so polovica ali četrtina navedene vrednosti. Hitrost vzorčenja je zelo dobro definirana, saj je pod nadzorom kristalnega oscilatorja.

Zvočna kartica vzorči z ločljivostjo 16 bitov na kanal. Kvalitetne zvočne kartice dejansko vzorčijo s tako ločljivostjo (ENOB = 16), kar je zelo dobro. Drugim nagaja šum in motnje v notranjosti osebnega računalnika, zato je njihova dejanska ločljivost ENOB slabša.

Občutljivost zvočne kartice je ohlapno definirana, saj za zajemanje zvočnih signalov ta ni tako pomembna. Zato se ne moremo zanesti, da bomo z različnimi zvočnimi karticami isti signal zajeli tako, da bodo imeli izmerki enako amplitudo. Še več, tudi občutljivost iste zvočne kartice se spreminja s temperaturo in časom.

Zvočna kartica lahko signal tudi generira, pri tem veljajo enake pripombe kot za zajemanje. Dobro je določena frekvenca, slabše pa amplituda in ločljivost.

Zvočna kartica vsebuje lastni procesor, ki nadzira zajemanje in shranjevanje rezultatov v lastni začasni spomin. Delovanje tega procesorja nadziramo preko gonilnika, ki je zvočni kartici prilagojen in ga dobimo skupaj s kartico. Gonilnik vsebuje podprograme, s katerimi nastavimo način delovanja kartice in prenašamo podatke med kartico in podatkovnim spominom. Te podprograme kličemo iz našega

programa, ki je napisan v višjem programskem jeziku. Imena podprogramov so standardizirana, prav tako parametri, ki jih podprogramom posredujemo. V nadaljevanju sledijo navodila in zgledi za zajemanje in generiranje niza, na koncu je še zgled za kontinuirano zajemanje in generiranje.

10.1.2. Zajem niza izmerkov

10.1.3. Generiranje niza

10.1.4. Kontinuirano zajemanje in generiranje

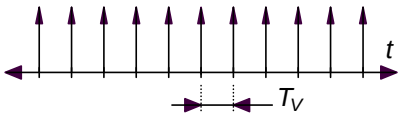
10.2. PCI vodilo – odvisno od periferije

11. Zajemanje podatkov

11.1. Vzorčeni signali

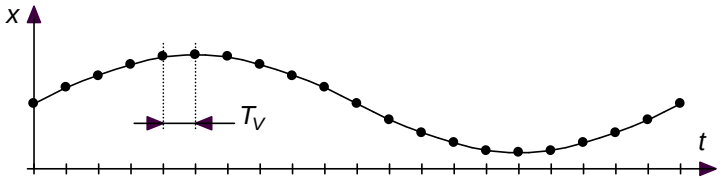
Najprej zapišimo zvezni analogni signal $x_a(t)$ in njegovo digitalno verzijo $x_d(kT_V)$, ki je diskretna v času (in velikosti, kar pa bomo za zdaj spregledali). Digitalni signal zapišemo v obliki neskončne vrste, časovni interval med zaporednimi izmerki je T_V. Dobimo ga, ko zvezno verzijo pomnožimo z neskončnim nizom delta funkcij, ki so med sabo razmaknjene za čas T_V po sliki 11.1.

$$x_d(kT_V) = x_a(t) \cdot \delta(t - 0 \cdot T_V) + x_a(t) \cdot \delta(t - 1 \cdot T_V) + x_a(t) \cdot \delta(t - 2 \cdot T_V) =$$
$$= x_a(t) \cdot \sum_k \delta(t - kT_V)$$



Slika 11.1: Niz delta funkcij, razmaknjenih za čas T_V.

Frekvenca vzorčenja f_V je recipročna vrednost časa T_V.



Slika 11.2: Analogni signal $x_a(t)$ vzorčimo in dobimo digitalni signal $x_d(kT_V)$, ki je na sliki označen s pikami

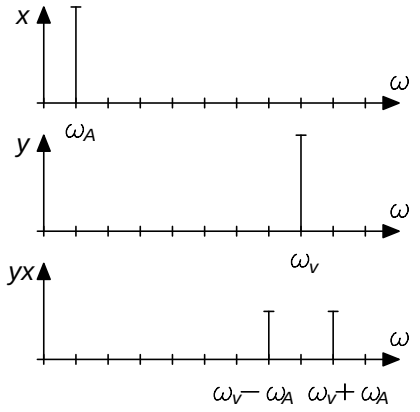
11.2. Hitrost vzorčenja – Nyquistov kriterij

S sistemom za zajem podatkov periodično vzorčimo analogni, časovno zvezni signal. Ker zvezni signal vzorčimo le v enakomernih časovnih intervalih, zavržemo

velik del informacije, ki jo vsebuje. Kako pogosto moramo vzorčiti, da bo mogoče vzorčeno, v času diskretno verzijo signala verno preslikati v prvotno zvezno obliko?

Diskretizirana verzija analognega signala je enaka produktu zveznega signala in niza delta funkcij. Za nazornejšo obrazložitev najprej poskusimo s harmoničnim vhodnim signalom *x*, frekvenca naj bo ω_A, ta signal množimo z drugim harmoničnim signalom *y* s frekvenco ω_V. Dobimo:

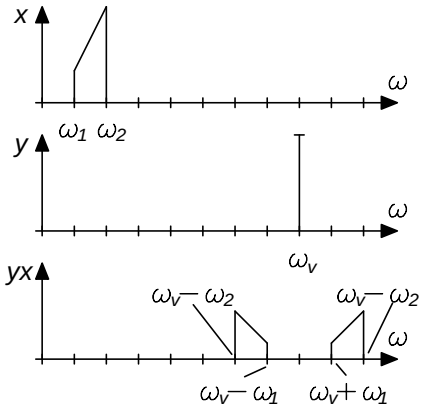
$$z = y \cdot x = \cos\omega_V t \cdot \cos\omega_A t = \frac{1}{2}[\cos(\omega_V + \omega_A)t + \cos(\omega_V - \omega_A)t]$$



Slika 11.3: Frekvenčni spekter produkta dveh harmonskih signalov

Na sliki 11.3 zgoraj je frekvenčni spekter signala *x*, na sredini spekter signala *y* in spodaj spekter produkta obeh signalov.

Vzemimo sedaj, da je vhodni signal *x* sestavljen iz množice harmonskih signalov od frekvence ω_1 do ω_2, pri tem naj imajo zaradi nazornosti razlage komponente pri višjih frekvencah večjo amplitudo. Spekter takega signala je na sliki 11.4 zgoraj. Ko tak signal pomnožimo s harmonskim signalom s frekvenco ω_V na sliki v sredini, dobimo spekter produkta signalov na isti sliki spodaj. To je hkrati tudi značilen spekter amplitudno moduliranega signala.

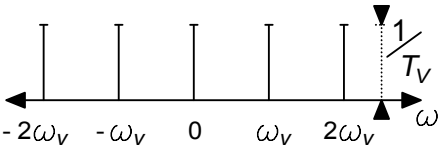


Slika 11.4: Frekvenčni spekter produkta harmoničnega signala in signala, ki zaseda pas frekvenc

Zanima nas frekvenčni spekter diskretizirane verzije zveznega signala. Množiti moramo torej spekter vhodnega zveznega signala in spekter niza delta funkcij s slike 11.1, ki ga izračunamo z razvojem v Fourierovo vrsto:

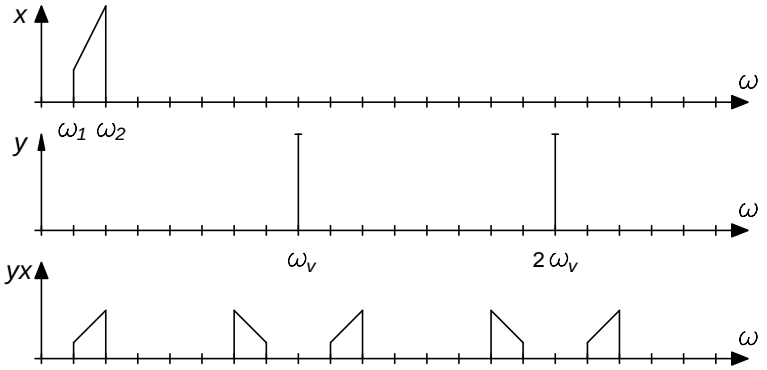
$$\mathcal{F}(\omega) = \frac{1}{T_V}\int_{-T_V/2}^{T_V/2} \delta(t)\, e^{-\frac{ik2\pi t}{T_v}}\, dt = \frac{1}{T_V}$$

Spekter niza delta funkcij je na sliki 11.5 in je periodičen s frekvenco ω_V. Ko ta spekter pomnožimo s spektrom vhodnega signala *x* s slike 11.4, dobimo spekter po sliki 11.6. Problemi se pojavijo, ko se frekvenčni spekter analognega signala razširi. Vzemimo, da

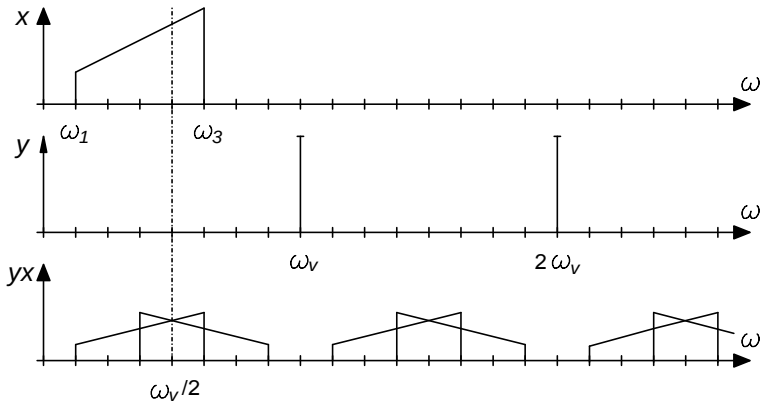


Slika 11.5: Frekvenčni spekter niza delta funkcij je periodičen

spekter analognega signala *x* sega do frekvence ω_3, ki je večja od $\omega_V/2$, ustrezna slika produkta vhodnih signalov je na sliki 11.7. Vidimo, da se deli spektrov prekrivajo. K amplitudi pri frekvenci ω_3 prispevata dve komponenti, tista zaradi amplitude zveznega signala pri frekvenci ω_3 in tista zaradi amplitude zveznega signala pri frekvenci $\omega_V - \omega_3$. Teh dveh komponent ni več mogoče ločiti in iz spektra rekonstruirati prvotnega signala, saj ne vemo, kolikšen delež je prispeval zvezni signal pri posamezni od obeh omenjenih frekvenc.



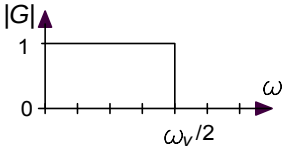
Slika 11.6: Spekter vzorčenega signala, ki je produkt spektra zveznega signala in spektra niza delta funkcij



Slika 11.7: Spekter vzorčenega signala, kadar je spekter zveznega signala širši od $\omega_V/2$

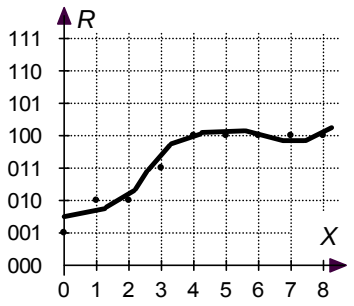
Do prekrivanja spektrov ne pride, če je zvezni signal frekvenčno omejen na največ $\omega_V/_2$, tej frekvenci pravimo Nyquistova frekvenca. Sistem za zajemanje analognih signalov moramo torej zasnovati tako, da frekvenca vhodnega signala ne preseže vrednosti $\omega_V/_2$. To storimo tako, da v modul za analogno obdelavo signala pred ADC vstavimo nizkoprepustni filter s prelomno frekvenco $\omega_V/_2$.

V idealnem primeru bi uporabili filter s frekvenčno karakteristiko po sliki 11.8. Takega v svetu analogne elektronike ne moremo narediti, možni so le približki. Če želimo zadostiti zgornji zahtevi, sme filter pri Nyquistovi frekvenci prepuščati le toliko signala, da ga ADC ne zazna, torej mora biti amplituda analognega signala manjša od 1 LSB (»Least Significant Bit«) pretvornika. Kadar imamo na primer opravka s pretvornikom z ločljivostjo 10 bitov, moramo amplitudo analognega signala pri Nyquistovi frekvenci $\omega_V/_2$ zmanjšati na eno tisočino vrednosti pri ostalih frekvencah, torej potrebujemo filter, ki pri Nyquistovi frekvenci duši za 60dB. Če filtriramo z navadnim RC členom, moramo postaviti prelomno frekvenco člena za tri dekade pred Nyquistovo frekvenco, kar je popolnoma nesprejemljivo, saj pokvarimo občutljivost pretvornika že daleč pred Nyquistovo frekvenco. Očitno je treba uporabiti analogne filtre višjega reda, delno sprejemljiv se zdi filter vsaj šestega reda, pri katerem dušenje narašča za 120dB na dekado in je zato mogoče postaviti prelomno frekvenco filtra približno pol dekade pred Nyquistovo frekvenco.



Slika 11.8: Želena amplitudna karakteristika nizkoprepustnega filtra pred ADCjem

Take analogne filtre je težko narediti, ker zahtevajo elektronske komponente z veliko točnostjo in stabilnostjo. Zaradi velikega napredka pri hitrosti vzorčenja in računanja je danes bolj enostavno uporabiti mnogo hitrejši ADC in procesor ter shajati z bolj enostavnimi filtri pred pretvornikom. Frekvenčni pas zajetega signala naknadno omejimo z digitalnim filtrom, ki je vgrajen v programsko opremo procesorja.



Slika 11.9: Digitalni signal po velikosti ni popolnoma enak analognemu zaradi napak kvantizacije. S pikami je označena digitalna verzija signala

Večja hitrost vzorčenja in naknadno digitalno filtriranje prinaša še eno ugodnost. Pri pretvorbi iz analogne v digitalno vrednost ADC zaokrožuje rezultat, saj pretvorjena digitalna vrednost na splošno ni popolnoma enakovredna analogni, slika 11.9. Napaka je statistične narave in je odvisna od trenutne vrednosti vhodnega signala, zato jo obravnavamo kot šum (šum

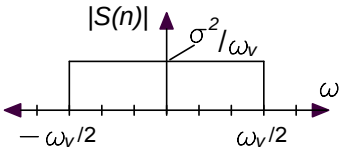
kvantizacije). Napaka je lahko velika do $\pm {}^{1}/_{2}$ LSB, vse vrednosti napake so enako verjetne. Efektivna vrednost šuma σ znaša:

$$\sigma^2 = \int_{-\frac{q}{2}}^{\frac{q}{2}} (x - \bar{x})^2 P(x) dx = {}^{q^2}/_{12} \quad , \quad \text{q==LSB velikost}$$

Iz tega izpeljemo SNR_{RMS}, razmerje moči koristnega signala *x(t)* proti moči šuma σ za idealni pretvornik z ločljivostjo *B* bitov:

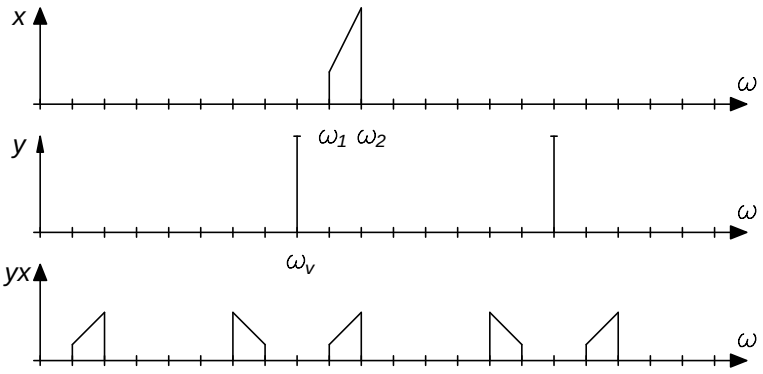
$$SNR_{RMS} = 10log\frac{x_{RMS}^2}{\sigma^2} = 6{,}02B + 4{,}77 - 20log\frac{A_{max}}{A}[dB]$$

Frekvenčni spekter šumne moči, ki je posledica napak kvantizacije, je podan na sliki 11.10 in je raven za vse frekvence od nič do Nyquistove. Če vzorčimo z veliko frekvenco, šum kvantizacije porazdelimo po širokem frekvenčnem območju. Ko med digitalno obdelavo signal filtriramo in mu omejimo frekvenčni pas, odstranimo del šuma, ki je posledica napak kvantizacije, to pa dejansko poveča ločljivost analogno digitalnega pretvornika. S prehitrim vzorčenjem lahko nadomestimo premajhno ločljivost pretvornika, tehniko, ki to omogoča, pa s tujko imenujemo »oversampling«.



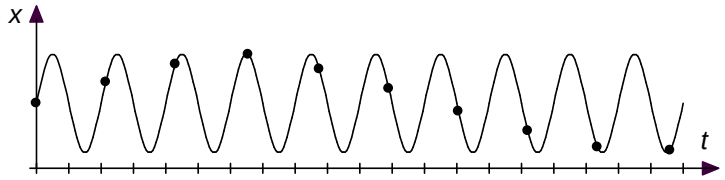
Slika 11.10: Šumna moč je enakomerno porazdeljena po frekvencah

Omeniti je treba, da je Nyquistova frekvenca izbrana zato, da se ognemo težavam, ki nastanejo zaradi prekrivanja delov spektra po vzorčenju. Če se znamo ogniti težavam tako, da omejimo frekvenčni pas vhodnega signala pred vzorčenjem, lahko s počasnim zajemanjem vzorčimo tudi signale s frekvenco, ki je nad Nyquistovo. Vzemimo za zgled, da je frekvenca vhodnega signala podana po sliki 11.11 in je večja od frekvence vzorčenja ω_V ter manjša od ${}^{3\omega_V}/_{2}$. Spekter signala po vzorčenju je podan na isti sliki spodaj in se ne razlikuje od spektra signala s slike 11.6, kjer je bil spekter vhodnega signala omejen na vrednost pod Nyquistovo frekvenco. Res smo



Slika 11.11: Spektri signala pri vzorčenju s frekvenco, ki je manjša od Nyquistove, pri tem je spekter vhodnega signala omejen.

tokrat izgubili podatek o absolutni vrednosti frekvence vhodnega signala, njegov spekter pa je pravilen in primeren za digitalno obdelavo. Zajemanje signala na ta način s tujko označujemo kot »undersampling«, časovni potek takega zajemanja je podan na sliki 11.12.



Slika 11.11: »Undersampling«, vztorčenje s frekvenco, ki je manjša od Nyquistove. Digitalni signal je označen s pikami, zvezni pa s polno črto.

11.3. Decimacija in interpolacija

Decimacija

Pogosto pride do tega, da je opazovani zvezni signal vzorčen s preveliko frekvenco. Morda do tega pride, ker uporabljamo tehniko »oversampling«, morda le zato, ker v naprej ne poznamo obsega frekvenc zveznega signala in ga želimo zajemati z nekaj varne rezerve.

Pri obdelavi digitaliziranih signalov je količina potrebnih matematičnih operacij v časovni enoti sorazmerna frekvenci vzorčenja. Če želimo manj operacij, vzorčimo bolj poredko, vendar spoštujemo Nyquistov kriterij. Dodaten vzrok za primerno frekvenco vzorčenja je digitalno filtriranje, za katerega so numerične operacije manj zahtevne, če so prelomne frekvence filtrov v srednjem delu Nyquistovega področja.

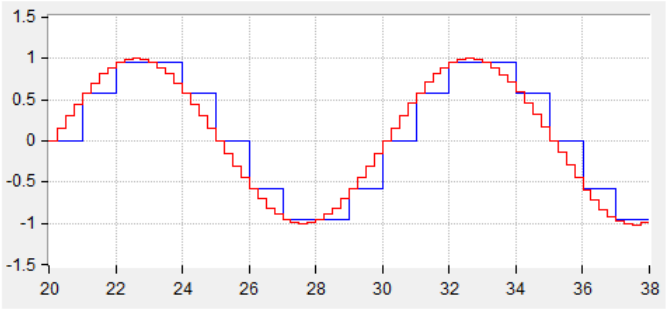
Frekvenco vzorčenja lahko po vzorčenju prepolovimo tako, da iz zajetih vzorcev izločimo vsaki drugi vzorec. Pravimo, da zajeti signal decimiramo s faktorjem dva. Predelani signal je enakovreden zajetemu, če le frekvenčni spekter zveznega signala sega največ do polovice Nyquistove frekvence. Rezultat je enak, kot da bi začetni zvezni signal vzorčili s polovično frekvenco.

Kadar je zvezni signal zelo majhne frekvence v primeri s frekvenco vzorčenja, lahko izločimo večji del zajetih vzorcev. Če sega največja frekvenca zveznega signala le do dvajsetine frekvence vzorčenja, je smiselno zajeti signal decimirati s faktorjem 10, obdržimo torej le vsak deseti vzorec, vse ostale zavržemo. Po taki decimaciji sega največja frekvenca zajetega signala natanko do Nyquistove meje, torej je zvezni signal vzorčen dovolj pogosto.

Interpolacija

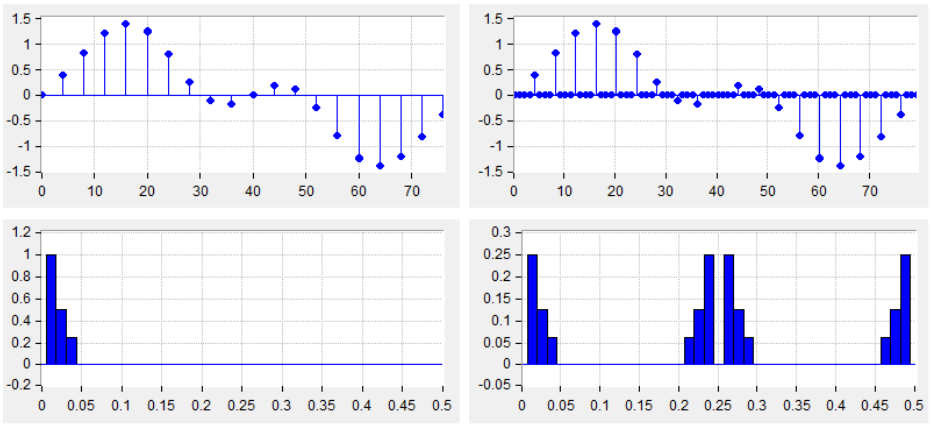
Če vzorčeni signal, ki sicer ustreza zahtevam Nyquista, pošljemo na DAC, bo generirani električni signal le grobo definiran, slika 11.12, modra sled. Takrat želimo

vzorčeni signal definirati v še več točkah, da bo generirana vrednost bolj primerne in gladke oblike, glej rdečo sled na isti sliki. Do sorodnih zahtev po večji hitrosti vzorčenja lahko pride tudi pri digitalnem filtriranju, saj so matematični postopki numerično manj zahtevni, če obdelujemo signale pri sredini Nyquistovega območja.



Slika 11.12: Harmonski signal, ki je generiran le desetkrat v periodi, je grob in stopničast. Ko isti signal definiramo večkrat v periodi, je oblika lažje prepoznavna. Obe osi diagrama sta normirani.

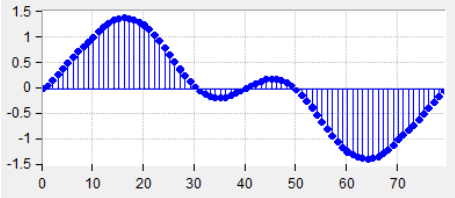
Postopku, ki med znane vzorce vrivamo dodatne tako, da je rezultirajoča krivulja čim bolj gladka, rečemo interpolacija. Iz matematike poznamo več metod za interpolacijo, omenimo naj le linearni približek in prilagajanje polinomov ali funkcij. Pri digitalni obdelavi podatkov so na voljo še druga, morda enostavnejša sredstva.



Slika 11.13: Interpolacija. Levo zgoraj je originalni digitalni signal, sestavljen iz treh harmonskih komponent različnih velikosti, levo spodaj je njegov frekvenčni spekter, preračunan na končno frekvenco vzorčenja. Desno zgoraj je isti digitalni signal z vstavljenimi tremi ničelnimi vzorci med vsak par vzorcev iz originalnega signala, desno spodaj pa je spekter modificiranega signala. Horizontalna os za spektre je podana kot razmerje f/f_v, ostale osi so normirane.

Interpoliramo tako, da med par sosednjih vzorcev zajetega signala najprej vstavimo nove vzorce, ki imajo vrednost nič. Če vstavimo en sam vzorec, govorimo o interpolaciji s faktorjem dva, frekvenca vzorčenja se v tem primeru podvoji. Dodani ničelni vzorci spremenijo frekvenčni spekter signala, slika 11.13 (za interpolacijo s faktorjem štiri so vrinjeni trije ničelni vzorci, frekvenca vzorčenja se početveri). Dodatne komponente ne spremenijo (položaja in velikosti) originalnih komponent, saj so pri drugih frekvencah. Ker dodatne komponente spremenijo obliko signala, jih ne potrebujemo, zato jih odstranimo z nizkoprepustnim filtrom, ki ima prelomno frekvenco pri $0.1f_v$.

Celoten postopek vstavljanja ničelnih vzorcev in odstranjevanja neželjenih komponent imenujemo interpolacija. Postopek zmanjša amplitudo rezultirajočega signala, glej spekter spodaj desno, slika 11.13, kar nadoknadimo z množenjem signala s štiri. Na sliki 11.14 je končni, interpolirani signal.



Slika 11.14: Interpolirani signal

11.4. Matematična analiza spektra diskretnega signala

Zaradi zajemanja spekter diskretnega signala ni enak spektru zveznega signala, kar smo nazorno pokazali v 11.2. Uporabimo oznake iz poglavja 11.1 in izračunajmo spekter diskretne verzije signala. Ta je dan z:

$$\mathcal{F}(x_d) = \int_{-\infty}^{\infty} x_d(kT_v) \cdot e^{-i\omega t}\, dt = \int_{-\infty}^{\infty} x_a(t) \cdot \left(\sum_{k=-\infty}^{\infty} \delta(t - kT_v) \right) \cdot e^{-i\omega t} dt$$

Del v oklepaju predstavlja niz delta sunkov *nizδ*, za te smo že izračunali spekter, ki je sestavljen in neskončnega niza harmoničnih komponent z velikostjo $1/T_v$, razmaknjenih za ω_v. Zato lahko tak niz zapišemo tudi z množico harmonskih nihanj:

$$niz\delta = \frac{1}{T_v} \sum_{k=-\infty}^{\infty} e^{ik\omega_v t}$$

Zato zgornji integral zapišemo:

$$\mathcal{F}(x_d) = \int_{-\infty}^{\infty} x_a(t) \cdot \left(\frac{1}{T_v} \sum_{k=-\infty}^{\infty} e^{ik\omega_v t} \right) \cdot e^{-i\omega t} dt$$

Po zamenjavi vrstnega reda integriranja in seštevanja dobimo:

$$\mathcal{F}(x_d) = \frac{1}{T_v} \sum_{k=-\infty}^{\infty} \int_{-\infty}^{\infty} x_a(t) \cdot e^{-i\Omega t} dt$$

Pri tem velja: $\Omega = \omega - k\omega_v$. Izraz pod znakom sumacije je natanko spekter zveznega signala, le parameter Ω je nov. Spekter diskretiziranega signala je sestavljen iz utežene množice spektrov zveznega signala, vsak od teh je premaknjen za ω_v proti ostalim. To pa je natanko rezultat, do katerega smo se s sklepanjem pririnili v poglavju 11.2.

Opozoriti velja na utež $1/T_v = f_v$. Zveznemu signalu lahko pripišemo energijo ves čas, diskretni signal pa prenaša energijo le občasno. Vidimo, da prenašana energija narašča s frekvenco vzorčenja f_v. Ko bomo v naslednjih poglavjih zvezni spekter filtrov predelovali v diskretno obliko, bomo morali to utež upoštevati.

12. Filtriranje signalov - FIR

12.1. Prevajanje signalov skozi linearni digitalni sistem

Lastnosti linearnih sistemov, ki jih uporabimo v tem zapisu, so:

e) Vzročnost: Izhodni signal je posledica vhodnega signala, kadar je vhodni signal enak nič in vsi prehodni pojavi izzvenijo, je izhodni signal konstanten. Ker je izhodni signal posledica vhodnega signala, je izhodni signal vedno zakasnjen za vhodnim signalom.

f) Homogenost: če vhodni signal *x* povzroči na izhodu signal *y*, potem *Ax* povzroči *Ay*. Pri tem je *A* velikost vhodnega signala.

g) Aditivnost: če vhodni signal *x*' povzroči izhodni signal *y*' in vhodni signal *x*'' povzroči izhodni signal *y*'', potem vsota vhodnih signalov *x*' in *x*'' povzroči izhodni signal, ki je enak *y*' + *y*''

h) Časovna neodvisnost: lastnosti sistema se s časom ne spreminjajo. Če vhodni signal *x*' povzroči izhodni signal *y*' ob prvi meritvi, bo enak vhodni signal povzročil enak izhodni signal ne glede na to, kdaj bomo meritev ponovili.

Linearni analogni sistem popišemo s prenosno funkcijo *T(s)*, ki je enaka razmerju Laplace-ovih transformov izhodnega in vhodnega signala.

$$T(s) = \frac{\mathcal{L}(y(t))}{\mathcal{L}(x(t))}$$

Pri tem je Laplace-ov transform vhodnega signala *x(t)* podan z:

$$\mathcal{L}\big(x(t)\big) = \int_{-\infty}^{\infty} x(t)e^{-st}dt$$

Linearne sisteme lahko vzbujamo z značilnimi signali, kot so na primer delta sunek $\delta(t)$, odziv nanj imenujemo $h(t)$ in napetostna stopnica $u(t)$, odziv nanjo je $w(t)$. Ker je:

$$\mathcal{L}\big(\delta(t)\big) = \int_{-\infty}^{\infty} \delta(t)e^{-st}dt = 1$$

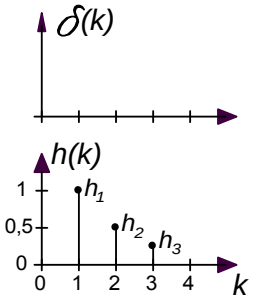
je pri vzbujanju z delta sunkom prenosna funkcija sistema kar enaka Laplace-ovemu transformu *h(s)* izhodnega signala *h(t)*, slika 12.1.

Neznani sistem testiramo tako, da na njegov vhod priključimo delta sunek in opazujemo izhodni signal *h(t)*. Ker v elektroniki ne moremo generirati delta funkcije, ki bi bila neskončno velika in hkrati neskončno ozka, uporabimo približek: delta funkcija mora trajati mnogo manj časa, kot znašajo značilne časovne konstante v testiranem sistemu. Če nas zanima prenosna funkcija sistema *T(s)*, iz zajetega *h(t)* izračunamo Laplace-ov transform *h(s)*.

Slika 12.1: Odziv analognega sistema na delta sunek

Ko poznamo odziv sistema na delta funkcijo $h(t)$, lahko z upoštevanjem linearnosti sistema izračunamo odziv na katerikoli vhodni signal. Vhodni signal *x(t)* razstavimo na niz zakasnjenih delta sunkov, vsakemu od njih pa pripišemo amplitudo tako, da skupaj sestavljajo vhodni signal *x(t)*. Za vsakega od delta sunkov v nizu izračunamo delni odziv sistema in vse delne sisteme med sabo seštejemo da dobimo odziv na vhodni vzbujalni signal *x(t)*.

Za zgled poglejmo, kako izračunati odziv digitalnega sistema na vzbujanje. Vzemimo, da je odziv na delta sunek podan kot $h(k) = \{1, 0{,}5, 0{,}25\}$, pri tem indeks *k* teče od 1 do 3, časovni interval med zaporednimi vzorci pa je konstanten. Odziv na vzbujanje z delta sunkom je na sliki 12.2.



Slika 12.2:Zgled za impulzni odziv h(k)

Vhodni signal je na sliki 12.3. Naj bo sestavljen iz niza $x(k) = \{1, 3, 2, 2\}$, pri tem indeks *k* teče od 0 do 3. Vhodni signal s komponento x_0 povzroči odziv y_a, s komponento x_1 pa odziv y_b ter tako naprej do zadnje komponente x_3. Za določanje izhodnega signala moramo sešteti istoležne delne odzive y_a do y_d pri posameznih indeksih, ta za indeks tri na primer znaša:

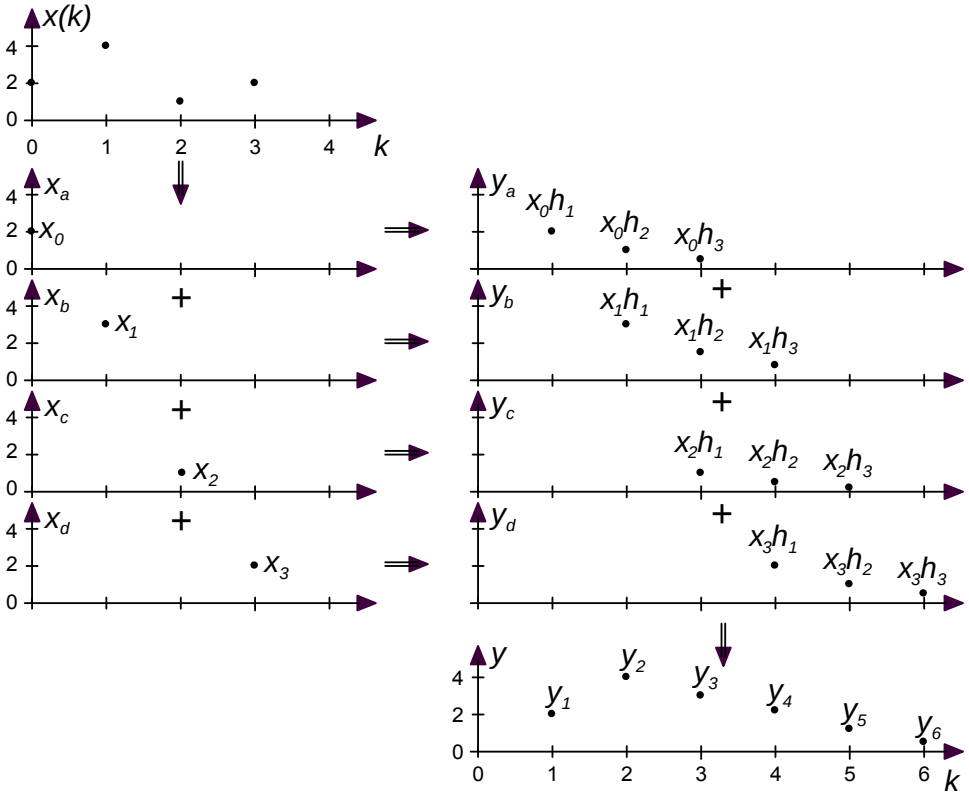
$$y_3 = h_3x_0 + h_2x_1 + h_1x_2 = \sum_{m=1}^{3} h_m x_{3-m}$$

Za element y_k odziva posplošimo izraz v:

$$y_k = \sum_{m=1}^{M} h_m x_{k-m} \qquad \Rightarrow \qquad y(k) = h(k) * x(k)$$

Pri tem *M* označuje število elementov v odzivu na delta sunek. Odziv sistema je enak konvoluciji vhodnega signala in odziva na delta sunek $h(k)$.

Če v računalniku uporabimo formulo za konvolucijo, lahko z njim oponašamo katerikoli analogni ali digitalni sistem za obdelavo signala. Poznati in digitalizirati



Slika 12.3: Konvolucija vhodnega signala in impulznega odziva h(k) določa izhodni signal

moramo le odziv oponašanega sistema na delta sunek. Kadar želimo z računalnikom oponašati na primer analogni filter, moramo v konvolucijski formuli uporabiti enak odziv na delta sunek, kot ga ima oponašani analogni filter. To storimo tako, da na vhod analognega filtra dovedemo delta sunek in odziv filtra na sunek digitaliziramo. Problem sestavljanja digitalnih filtrov, ki računajo odziv na podlagi konvolucije, se zato reducira na izbiro primerne prenosne funkcije *T(s)* analognega filtra in digitalizacije odziva tega filtra na delta sunek *h(k)*.

12.2. Filtriranje FIR

Računanju odziva linearnega sistema s pomočjo konvolucije pravimo tudi filtriranje FIR (»Finite Impulse Response«). Ker je v odzivu na delta sunek le končno število elementov *h(k)*, odziv na delta izzveni najkasneje po *K* časovnih intervalih, pri tem je *K* enak številu elementov v *h(k)*.

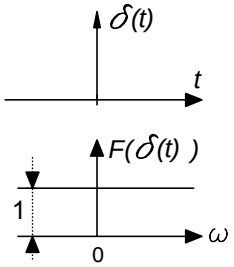
Za filtriranje FIR, ki ima željeno frekvenčno karakteristiko $T(i\omega)$, moramo izbrati pravi odziv na delta sunek *h(k)*. Frekvenčna karakteristike $T(i\omega)$ je definirana kot

razmerje Fourierovih transformov izhodne (*y(t)*) in vhodne (*x(t)*) napetosti opazovanega sistema:

$$T(i\omega) = \frac{\mathcal{F}(y(t))}{\mathcal{F}(x(t))}$$

Če na vhod sistema priključimo delta sunek po sliki 12.4 zgoraj, je odziv sistema enak *h(t)*. Fourierov transform delta sunka je enak:

$$\mathcal{F}\big(\delta(t)\big) = \int_{-\infty}^{\infty} \delta(t)\, e^{-i\omega t} dt = 1$$



Slika 12.4: Delta sunek in njegov frekvenčni spekter

Amplitude harmonskih komponent, ki sestavljajo delta sunek, so torej enake pri vseh frekvencah, frekvenčni spekter delta sunka je na sliki 12.4 spodaj. Pravimo, da je spekter bel. Iz povedanega zapišemo:

$$T(i\omega) = \mathcal{F}(h(t))$$

Od tod dobimo:

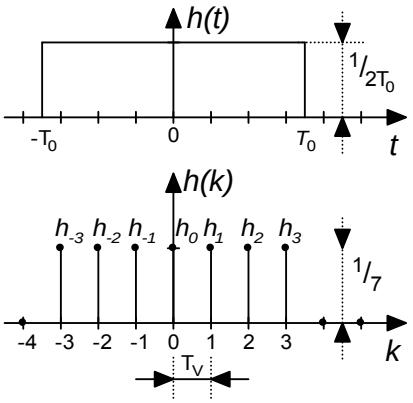
$$h(t) = \mathcal{F}^{-1}[\,T(i\omega)] \;= \frac{1}{2\pi} \int_{-\infty}^{\infty} T(i\omega) e^{i\omega t} d\omega$$

Funkcijo *h(t)* diskretiziramo in dobimo *h(k)*, ki jo uporabimo v formuli za konvolucijo. Digitalni sistem, ki računa po tej formuli, ima frekvenčno karakteristiko *T(iω)*.

12.2.1. Zgled: Frekvenčna karakteristika za bločno povprečenje

Pri bločnem povprečenju izračunamo povprečno vrednost *K* zaporednih izmerkov. Ko uporabimo formulo za konvolucijo, imajo vsi koeficienti *h(k)* enako vrednost $h(k) = {}^{1}/_{K}$ (slika 12.5 spodaj), pri tem je K število koeficientov, ki je navadno liho. Računanje pospešimo, če najprej izračunamo vsoto *K* zaporednih členov, nato pa vsoto delimo s *K*.

V analognem svetu bločno povprečenje zaznamuje odziv na delta sunek *h(t)*, ki je konstanten v časovnem intervalu od $-T_0$ do $+T_0$, slika 12.5 zgoraj in znaša $h(t) = {}^{1}/_{2T_0}$. Tak odziv prevedemo v diskretno obliko tako, da upoštevamo čas med dvema zaporednima vzorcema T_V, kar je na sliki nakazano s črtkanimi črtami. Število koeficientov v diskretni verziji *h(k)*



Slika 12.5: Bločno povprečenje, impulzni odziv za analogni in digitalni filter

je zato enako:

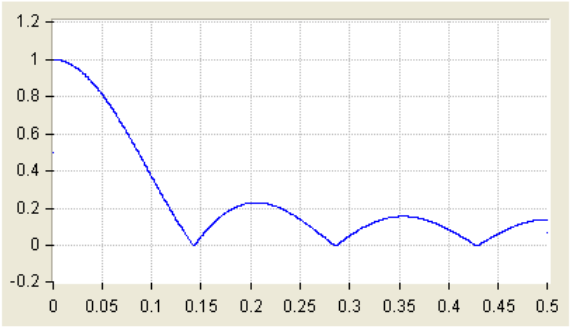
$$\mathrm{K} = 2 \cdot {}^{T_0}/_{T_V}$$

Ker je frekvenčna karakteristika filtra *T(iω)* podana s formulo zgoraj, dobimo:

$$T(i\omega) = \mathcal{F}\big(h(t)\big) = \int_{-\infty}^{\infty} h(t) e^{-i\omega t} dt = \frac{\sin \omega T_0}{\omega T_0}$$

Potek ojačenja filtra sledi funkciji ${}^{\sin x}/_{x}$, njegova absolutna vrednost je na sliki 12.6. Ojačenje je največje (1) za enosmerne signale in enako nič za signale s frekvenco f_0 ($f_V = {}^{1}/_{T_V}$):

$$f_0 = f_v \frac{n}{K}$$

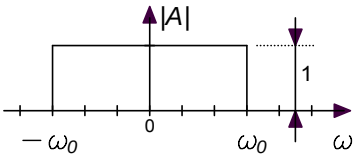


Slika 12.6: Amplitudna karakteristika za bločno povprečenje po sliki 12.5, abscisna os je f/f_v

Pri tem je n naravno število. Bločno povprečenje je torej zelo primerno za izločanje motilnega signala s točno določeno frekvenco in še bolj za izločanje na primer pravokotnega signala, ki ima komponente pri lihih mnogokratnikih osnovne frekvence. Ker je prehod med prepustnim in zapornim delom frekvenčne karakteristike blag in prepušča vhodni signal še pri višjih frekvencah nad f_0, ni najprimernejši za nizkoprepustni filter.

12.2.2. Zgled: koeficienti *h(k)* za idealni nizkoprepustni filter

Frekvenčna karakteristika za idealni nizkoprepustni filter je na sliki 12.7. Ojačenje je ena za pas frekvenc od nič do mejne frekvence ω_0, od tu naprej je ojačenje enako nič.



Slika 12.7: Frekvenčna karakteristika za idealni nizkoprepustni filter

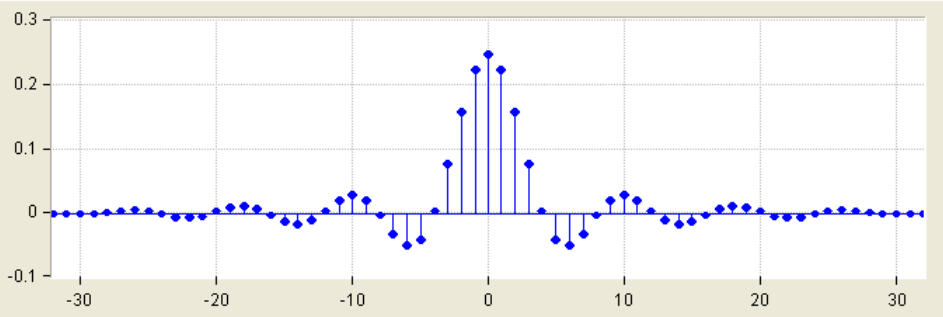
Določimo potrebni odziv *h(t)* na delta sunek, ki zagotavlja želeno frekvenčno karakteristiko *T(iω)*. Računamo po formuli iz poglavja 12.2:

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} T(i\omega) e^{i\omega t} d\omega = \frac{\omega_0}{\pi} \frac{\sin \omega_0 t}{\omega_0 t}$$

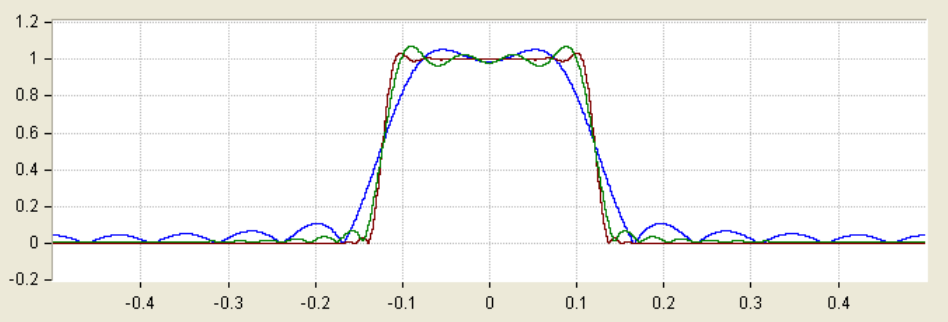
V diskretni verziji nadomestimo čas t s kT_V, pri tem je T_V čas med dvema zaporednima vzorcema, k pa teče od –∞ do +∞. Upoštevamo še, da je treba izenačiti amplitudi tako predelanega spektra po 11.4 in dobimo:

$$h(k) = 2\,\frac{f_0}{f_v}\,\frac{\sin 2\pi k\,\frac{f_0}{f_V}}{2\pi k\,\frac{f_0}{f_V}}$$

Na sliki 12.8 so za zgled narisane vrednosti nekaterih koeficientov *h(k)*, ki jih moramo vgraditi v konvolucijsko formulo, da dobimo idealni nizkoprepustni filter z mejno frekvenco f_0. Koeficientov bi moralo biti neskončno mnogo, česar iz praktičnih razlogov ne moremo dovoliti, saj bi računanje konvolucije trajalo neskončno dolgo. Zato število koeficientov omejimo. Uporabimo le srednji koeficient h_0 in enako število koeficientov na njegovi levi in desni, to pa vpliva na frekvenčno karakteristiko filtra. Na sliki 12.9 je za zgled podana frekvenčna karakteristika filtra za primer, ko število filtrskih koeficientov omejimo.



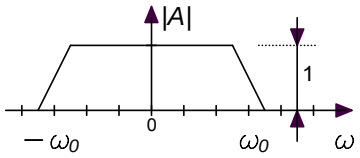
Slika 12.8: Koeficienti za idealni nizkoprepustni filter, ordinatna je brez dimenzije, abscisna predstavlja k



Slika 12.9: Frekvenčna karakteristika nizkoprepustnega filtra z omejenim številom koeficientov, 13 (modra),31 (zelena),63 (rjava), abscisna os je f_0/f_v, ordinata pa amplituda odziva

Z omejitvijo števila koeficientov torej poslabšamo amplitudno karakteristiko. Ob prehodu iz prepustnega v zaporni pas je karakteristika bolj položna, pa še dušenje v zapornem delu karakteristike je pri manjšem številu koeficientov slabše. Težave lahko omilimo na dva načina:

a) Že na začetku načrtovanja zahtevamo blažji prehod med zapornim in prepustnim delom amplitudne karakteristike po sliki 12.10.
b) Po omejitvi števila koeficientov njihovo velikost še korigiramo s tako imenovano okensko funkcijo.

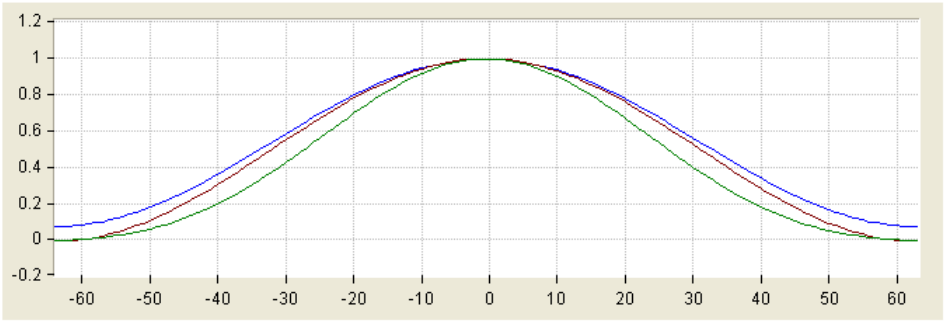


Slika 12.10: Milejša oblika frekvenčne karakteristike za nizkoprepustni filter

12.3. Uporaba okenskih funkcij

Z okensko funkcijo korigiramo vrednosti koeficientov v odzivu na delta sunek. Gre za to, da vrednosti koeficientov ob odrezanem delu zlagoma privedemo do vrednosti nič. To dosežemo z množenjem koeficientov s funkcijo, ki se ob meji spusti do minimalne vrednosti, v sredini področja pri koeficientu z indeksom nič pa ostanejo vrednosti koeficientov skoraj nespremenjene.

Okenske funkcije so predlagali različni avtorji, najbolj znani so Hanning, Hamming Blackman in Kaiser. Podrobnosti so podane v literaturi %%, tu navedimo le nekaj standardnih formul za okenske funkcije, grafi so na sliki 12.11.



Slika 12.11: Okenske funkcije: Hanning(rjava), Hamming(modra), Kaiser (zelena)

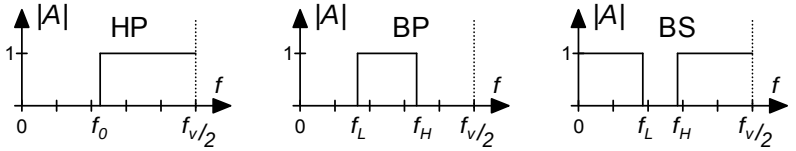
Avtor	Formula, $\lvert k\rvert \leq {}^{(K-1)}/_{2}$
Hanning	$0{,}5 + 0{,}5 \cdot \cos\left(\frac{2\pi k}{K}\right)$
Hamming	$0{,}54 + 0{,}46 \cdot \cos\left(\frac{2\pi k}{K}\right)$
Blackman	$0{,}42 + 0{,}5 \cdot \cos\left(\frac{2\pi k}{K-1}\right) + 0{,}08 \cdot \cos\left(\frac{4\pi k}{K-1}\right)$

Kaiser	$\dfrac{\mathrm{I}_0\left(\beta\left\{1-\left[\dfrac{2k}{(K-1)^2}\right]\right\}^{1/2}\right)}{I_0(\beta)}$

V zadnji vrstici je $I_0(x)$ modificirana Besselova funkcija ničtega reda, parameter β pa definira gladkost prehoda med prepustnim in zapornim delom filtrove karakteristike ter najmanjše dušenje v zapornem delu. Za β=0 imamo opravka s pravokotnim oknom, za vrednost β=5,44 pa je okno zelo podobno Hammingovemu oknu. Večja, ko je vrednost β, boljše je dušenje v zapornem delu, a na račun položnejšega prehoda med prepustnim in zapornim delom.

12.4. Druge karakteristike filtrov FIR

Na podoben način z obratno fourierovo transformacijo iz poglavja 12.2 določimo tudi filtrske koeficiente za filtre visoko-prepustnega tipa, pasovno-prepustnega in pasovno-zapornega tipa, slika 12.12. Gre pa tudi enostavneje.



Slika 12.12: Karakteristike visoko-prepustnega (HP), pasovno-prepustnega (BP) in pasovno-zapornega (BS) filtra

Idealni visoko prepustni filter (HP) ima karakteristiko, ki jo dobimo, če od karakteristike filtra, ki prepušča vse frekvence, odštejemo karakteristiko nizkoprepustnega filtra. Zato moramo med sabo odšteti vrednosti ustreznih filterskih koeficientov:

$$h(0) = 1 - 2\,\frac{f_0}{f_v} \qquad h(k) = -2\,\frac{f_0}{f_v}\,\frac{\sin 2\pi k\dfrac{f_0}{f_V}}{2\pi k\dfrac{f_0}{f_V}}$$

Karakteristiko idealnega pasovno prepustnega filtra (BP) dobimo, če izračunamo razliko med karakteristikama dveh nizkoprepustnih filtrov s primernima mejnima frekvencama, kar da koeficiente:

$$h(0) = 2\cdot\frac{f_H - f_L}{f_v} \qquad h(k) = 2\,\frac{f_H}{f_v}\,\frac{\sin 2\pi k\dfrac{f_H}{f_V}}{2\pi k\dfrac{f_H}{f_V}} - 2\,\frac{f_L}{f_v}\,\frac{\sin 2\pi k\dfrac{f_L}{f_V}}{2\pi k\dfrac{f_L}{f_V}}$$

Karakteristiko idealnega pasovno zapornega filtra (BS) dobimo, če od karakteristike filtra, ki prepušča vse frekvence, odštejemo karakteristiko idealnega pasovno prepustnega filtra. To da koeficiente:

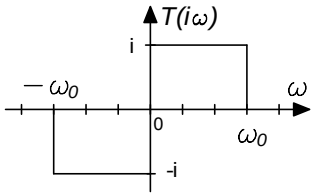
$$h(0) = 1 - 2 \cdot \frac{f_H - f_L}{f_v} \qquad h(k) = 2\,\frac{f_L}{f_v}\,\frac{\sin 2\pi k \frac{f_L}{f_V}}{2\pi k \frac{f_L}{f_V}} - 2\,\frac{f_H}{f_v}\,\frac{\sin 2\pi k \frac{f_H}{f_V}}{2\pi k \frac{f_H}{f_V}}$$

Pri tem sta frekvenci f_L in f_H mejni frekvenci filtra. Ker je koeficientov neskončno mnogo, moramo njihovo število omejiti, zaradi zmanjšanja valovitosti karakteristike pa še prilagoditi njihovo velikost z okensko funkcijo.

12.5. Hilbertova transformacija

Pogosto potrebujemo signal, ki je za originalom zakasnjen za $\pi/2$. Zakasnjena verzija signala na primer olajšuje določanje trenutne amplitude harmoničnega signala. Treba je le zajemati signal, katerega trenutna amplituda nas zanima, ga digitalno zakasniti za 90 stopinj, in sproti računati kvadratni koren iz vsote kvadrata trenutne vrednosti zajetega signala in kvadrata zakasnjene verzije istega signala.

Prenosna funkcija *T(iω)*za kasnilni filter ima konstantno velikost v vsem željenem frekvenčnem področju od $-\omega_0$ do $+\omega_0$, fazna karakteristika pa zahteva kot $+\pi/2$ za pozitivne frekvence in $-\pi/2$ za negativne frekvence, slika 12.13. Taka izbira zagotavlja realne koeficiente FIR filtra.



Slika 12.13: T(iω) za Hilbertov transfom

Koeficiente filtra določimo po postopku iz poglavja 12.2:

$$h(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} T(i\omega)e^{i\omega t}d\omega = \frac{2\omega_0}{2\pi}\,\frac{-1+\cos(\omega_0 t)}{\omega_0 t}$$

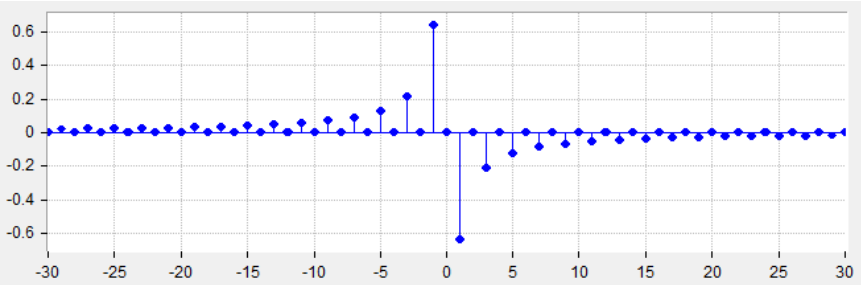
To funkcijo diskretiziramo tako, da t zamenjamo s kT_v ter upoštevamo utež. Rezultat je:

$$h(k) = 2\,\frac{f_0}{f_v}\,\frac{-1+\cos 2\pi k\frac{f_0}{f_V}}{2\pi k\frac{f_0}{f_V}}$$

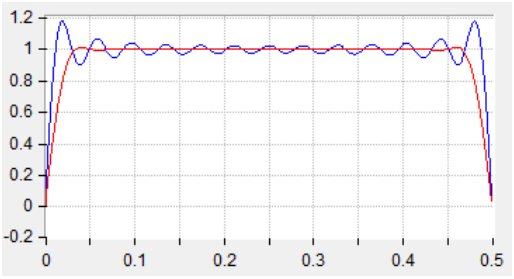
Pri tem *k* teče od -∞ do +∞. Če zahtevamo fazni premik 90 stopinj preko vsega območja, ki ga dopušča diskretiziranje ($f_0 = f_v/2$), se izraz poenostavi v (slika 12.14):

$$h(k) = \frac{-1+(-1)^k}{\pi k}$$

Ker za realizacijo filtra potrebujemo neskončno mnogo koeficientov, ga poenostavimo podobno, kot smo običajne FIR filtre: število koeficientov omejimo na razumno vrednost. To povzroči napake pri amplitudi in fazi izhodnega signala, slika 12.15, ki jih omilimo z okenskimi funkcijami. Do največjih napak (valovitost amplitude) prihaja pri mejah frekvenčnega območja.



Slika 12.14: Koeficienti Hilbertovega transforma za poln frekvenčni obseg



Slika 12.15: Amplituda odziva za zgornje filtersko jedro: brez okna (modro) in z uporabo okna (rdeče); abscisna os je normirana $f\,/\,f_v$

13. Filtriranje podatkov - IIR

13.1. Rekurzivne formule

Digitalizirane signale lahko filtriramo s pomočjo konvolucije, poznati moramo le vrednosti koeficientov v konvolucijski formuli, kar smo opisali v prejšnjem poglavju.

Filtriranje s konvolucijo je počasno, ker je koeficientov filterskega jedra za filtre z ostro definiranim prehodom med prepustnim in zapornim pasom veliko. Pričakujemo, da bi bilo računanja manj, če bi rezultat ne temeljil le na uteženi vsoti vzorcev vhodnega signala (običajna konvolucija), ampak tudi na predhodnih rezultatih. Želimo se dokopati do rekurzivne formule, s katero bi izkoristili prej izračunane rezultate in tako z manj množenji prišli do filtrov z enakovrednimi frekvenčnimi karakteristikami, kar bomo počeli v tem poglavju.

Pri bločnem povprečenju, na primer, se zdi nesmiselno, da za vsak novi rezultat y_k seštevamo niz M predhodnih izmerkov. Enako bi lahko od prejšnjega rezultata y_{k-1} odšteli najstarejši izmerek in prišteli novega. Namesto:

$$y_k = \sum_{m=0}^{M-1} x_{k-m}$$

Bi pisali:

$$y_k = y_{k-1} - x_{k-M+1} + x_k$$

V splošnem rekurzivno formulo za filtriranje zapišemo tako, da je rezultat y_k v k-tem koraku vsota uteženih vhodnih vzorcev in uteženih predhodnih rezultatov:

$$\begin{aligned} y_k = a_0 x_k + \quad & a_1 x_{k-1} + a_2 x_{k-2} + \cdots \\ & - (\ b_1 y_{k-1} + b_2 y_{k-2} + \cdots) \end{aligned}$$

Pri tem uteži predstavljajo koeficienti a in b. Upamo, da lahko d primernimi utežmi dosežemo poljubno frekvenčno karakteristiko filtriranja.

Uteži, ki omogočajo enostavno filtriranje, lahko uganemo. Filter z eksponentnim pozabljanjem naredimo tako, da izberemo koeficienta $a_0 = 1 - \alpha$ in $b_1 = -\alpha$, pri tem je $0 \leq \alpha \leq 1$. Kadar je α blizu ena, rezultat temelji v glavnem na prejšnjem rezultatu in le malo na novem izmerku, naredili smo nizkoprepustni filter. Vzemimo, da je vrednost y v nekem trenutku enaka ena in spremenimo vrednost vhodnega

signala na nič. Kasnejši rezultati računanja rekurzivne formule z zgornjimi koeficienti so zato α, α^2, α^3,..., in α^N po *N* računanjih. Pri običajnem analognem filtru prvega reda izhodni signal pade na vrednost e^{-1} po času, ki je enak časovni konstanti τ. Če enačimo lastnosti digitalnega filtra z eksponentnim pozabljanjem in analognega filtra prvega reda, velja $e^{-1} = \alpha^N$, od tu določimo vrednost τ za ekvivalentna filtra kot $\tau = N \cdot T_v = {-T_v}/{\ln \alpha}$. Tabela 13.1 podaja vrednosti časovne konstante digitalnega filtra z eksponentnim pozabljanjem, za katerega naj po *N* vzorcih izhodna vrednost pade na ${1}/{e}$.

α	$\tau = N\,T_v$
0,5 = 1 – 1/2	*1,44T_V*
0,875 = 1 – 1/8	*7,48T_V*
0,984375 = 1 – 1/64	*63,5T_V*
0,9921875 = 1 - 1/128	*127,5T_V*
0,999023 = 1 - 1/1024	*1023,5T_V*

Tabela 13.1: Koeficient α in časovna konstanta filtra

Visokoprepustni filter naredimo tako, da izberemo koeficiente: $a_0 = \frac{1+\alpha}{2}$, $a_1 = -\frac{1+\alpha}{2}$ in $b_1 = -\alpha$.

Navedena filtra nimata ostrega prehoda med prepustnim in zapornim delom karakteristike, za to so potrebni filtri višjih redov. Poskusimo lahko s tabeliranimi koeficienti iz priloge 2. S temi koeficienti je mogoče implementirati filtre do šestega reda, pri teh je prepustni pas mnogo ostreje ločen od zapornega.

Če nam ni do receptov, se do vrednosti koeficientov dokopljemo s pomočjo z-transformacije, ki je opisana v nadaljevanju.

13.2. Z-transformacija

Pri linearni elektroniki določamo izhodne signale vezij s pomočjo diferencialnih enačb. Ker je reševanje le-teh kompliciranо, uporabimo operatorski zapis diferencialnih enačb ali Laplace-ovo transformacijo, ki diferencialne enačbe prevede v algebraične. Te so bistveno lažje rešljive.

Tudi pri signalih, ki so diskretni v času, za lažje računanje uvedemo transformacijo, ki jo imenujemo z-transformacija. Ta matematični pripomoček prevede algebraično vrsto, ki predstavlja signal, v algebraično formulo. S pomočjo z z-transformacijo izraženih formul definiramo prenosno funkcijo za sistem, kjer so signali diskretni.

Diskretni signal *x(k)* zapišemo kot zaporedje vzorcev v času:

$$x(k) = \{x_0, x_1, x_2, x_3 \ldots\}$$

Govorimo o vzročnih sistemih, kjer je odziv posledica vzbujanja in se torej ne more pojaviti pred vzbujanjem. Z-transformacija *X(z)* takega signala je definirana kot:

$$X(z) = \sum_{k=0}^{\infty} x_k z^{-k}$$

Pri tem je *z* kompleksna spremenljivka. Do te definicije pridemo preko Laplace-ove transformacije, ko v formuli zvezni signal x_a(t) zamenjamo z diskretiziranim $x_d(kT_V)$ ter v rezultatu transforma e^{sT_V} zamenjamo z *z*.

Zgled:

Tabelirano za nekaj značilnih signalov (Tabela 13.2):

Diskretna vrsta, $n \geq 0$	z-transform	Območje konvergence
$\delta(k)$	1	povsod
$1 - e^{-\alpha k}$	$\frac{z(1-e^{-\alpha})}{z^2 - z(1+e^{-\alpha}) + e^{-\alpha}}$	$\lvert z\rvert > e^{-\alpha}$
$\cos(\alpha k)$	$\frac{z(z-\cos\alpha)}{z^2 - 2c\cos\alpha + 1}$	$\lvert z\rvert > 1$
$\sin(\alpha k)$	$\frac{z\sin\alpha}{z^2 - 2c\cos\alpha + 1}$	$\lvert z\rvert > 1$
α^k	$\frac{z}{z-\alpha}$	$\lvert z\rvert > \alpha$

13.2.1. Obratna z-transformacija

Obratna z-transformacija iz danega z-transforma vrne originalni signal. Iz definicije z-transforma je očitno:

$$X(z) = \sum_{k=0}^{\infty} x_k z^{-k} = x(0)z^0 + x(1)z^{-1} + x(2)z^{-2} + x(3)z^{-3} + \cdots$$

Torej lahko za dani z-transform izluščimo vrednosti originalnega signala tako, da izpišemo uteži pri posamezni potenci z^{-k}. Kadar je z-transform podan v obliki kvocienta polinomov, najprej opravimo deljenje in potem izluščimo vrednosti originalnega signala.

Zgled:

$$X(z) = \frac{1}{1 - 0{,}5z^{-1}} = 1 + 0{,}5z^{-1} + 0{,}25z^{-2} + 0{,}125z^{-3} + \cdots$$

Zato originalni signal izpišemo kot:

$$x(k) = \{1, \quad 0{,}5, \quad 0{,}25, \quad 0{,}125, \ldots\} = 0{,}5^k$$

Do originalnega signala lahko pridemo tudi s tabelo, kakršna je podana zgoraj. Z-transform razstavimo na člene, ki jih je mogoče najti v tabelah, nato iz njih izpišemo ustrezne vrste in jih sestavimo v skupno vrsto. Matematično popolnejša pot vodi preko računanja kompleksnega integrala po zaključeni poti, kjer si pomagamo z

računanjem residuov. Tega tu ne bomo podrobneje popisovali, bralec naj poišče razlago v drugje.

13.2.2. Operator zakasnitve z^{-1}

Z-transform pomnožimo s faktorjem z^{-m}. Rezultat je:

$$X(z)z^{-m} = \sum_{k=0}^{\infty} x_k z^{-k} z^{-m}$$
$$= x(0)z^{0-m} + x(1)z^{-1-m} + x(2)z^{-2-m} + x(3)z^{-3-m} + \cdots$$

Če primerjamo zgornji izraz s tistim v poglavju 13.2.1 vidimo, da sta desni strani enačaja enaki v primeru, da v zgornji formuli zaporedje *x(k)* zakasnimo za *m* členov. Množenje z-transforma s faktorjem z^{-m} torej predstavlja zakasnitev signala za *m* vzorcev. Enako velja tudi v primeru, da je vrsta sestavljena iz končnega števila členov.

Če torej velja:

$$x(k) \quad \leftrightarrow \quad X(z)$$

Velja tudi:

$$x(k-m) \quad \leftrightarrow \quad z^{-m}X(z)$$

13.3. Prenosna funkcija in povezava z diferenčno enačbo

Prenosno funkcijo *H(z)* digitalnega sistema definiramo kot razmerje z-transformov izhodnega in vhodnega signala, kar je kvocient dveh polinomov.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_N z^{-N} + \cdots + a_0 z^0}{b_M z^{-M} + \cdots + b_0 z^0} = \frac{\sum_{n=0}^{N} a_n z^{-n}}{\sum_{m=0}^{M} b_m z^{-m}}$$

Navzkrižno množenje da:

$$\sum_{m=0}^{M} b_m Y(z) z^{-m} = \sum_{n=0}^{N} a_n X(z) z^{-n}$$

Zadnja dela $Y(z)z^{-m}$ in $X(z)z^{-n}$ v sumacijah sta natanko z-transformaciji zakasnjenih signalov *y(k)* in *x(k)*, zato v k-tem koraku lahko pišemo:

$$\sum_{m=0}^{M} b_m y_{k-m} = \sum_{n=0}^{N} a_n x_{k-n}$$

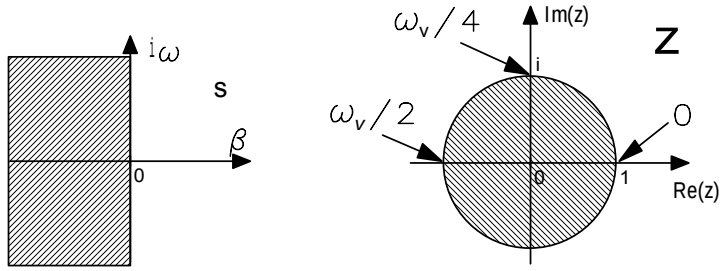
Iz tega sledi (za $b_0 = 1$):

$$y_k = \sum_{n=0}^{N} a_n x_{k-n} - \sum_{m=1}^{M} b_m y_{k-m}$$

Kar je natanko diferenčna enačba, do katere smo želeli v poglavju 13.1. Vsaki diferenčni enačbi torej lahko pripišemo prenosno funkcijo in obratno.

13.4. Področje stabilnosti in preslikava osi jω

Sistem, ki ga popisuje zvezna prenosna funkcija *T(s)* je stabilen, če je realni del vsakega pola prenosne funkcije negativen (Elektronika I). Mejo med deloma prostora $s = \beta + i\omega$ predstavlja abscisna os $i\omega$, pri tem gre frekvenca $-\infty < \omega < \infty$.



Slika 13.3: Poli v ravnini s za stabilni analogni sistem so levo od ordinatne osi, za stabilni digitalni sistem pa v notranjosti enotskega kroga v ravnini z.

Pri izpeljavi z-transformacije smo upoštevali, da je $z = e^{sT_V} = e^{\beta T_V} e^{i\omega T_V}$. Ordinatna os ($\beta = 0$) se v ravnini *z* prevede v vektor z dolžino $e^{\beta T_V} = 1$ ter kot, ki se s frekvenco enakomerno spreminja. Vrh vektorja popiše enotski krog v ravnini *z*, slika 13.3. Pri tem se frekvenca ω spreminja od 0 pri koordinati z{1,0} preko $\omega_V/_4$ pri koordinati z{0,i} do $\omega_V/_2$ pri koordinati z{-1,0}. Ker se v frekvenca lahko spreminja tudi v negativni smeri, lahko podobno sklepamo tudi za spodnjo polovico enotskega kroga. Pri povečevanju frekvence vhodnega signala preko $\omega_V/_2$ pride torej do dvoličnosti, ki botruje tudi Nyquistovi frekvenci; frekvenca vhodnega signala naj ostane pod $f_V/_2$.

Levi del ravnine *s* se prenese v notranjost enotskega kroga. Velja torej, da stabilen digitalni sistem popisuje prenosna funkcija *H(z)*, pri kateri so vsi poli v notranjosti enotskega kroga. Ničle diskretne prenosne funkcije so lahko kjerkoli v ravnini . Poli na enotskem krogu povzročijo periodično nihanje sistema, če je ta vsaj drugega reda.

13.5. Vrednotenje frekvenčne karakteristike diskretne prenosne funkcije

Frekvenčno karakteristiko analognega sistema, ki ga popisuje zvezna prenosna funkcija *T(s)* določimo tako, da spremenljivko *s* zamenjamo z $i\omega$ ter določimo absolutno vrednost dobljenega izraza (amplitudo) in njegov fazni kot.

Frekvenčno karakteristiko sistema, ki ga popisuje diskretna prenosna funkcija *H(z)* določimo tako, da spremenljivko *z* zamenjamo z $z = e^{sT_v} = e^{i\omega T_v}$ ter določimo absolutno vrednost izraza (amplituda) in fazni kot v odvisnosti od frekvence ω.

Zgled: Prenosna funkcija sistema za eksponentno pozabljanje je podana z:

$$H(z) = \frac{1-\alpha}{1-\alpha z^{-1}}$$

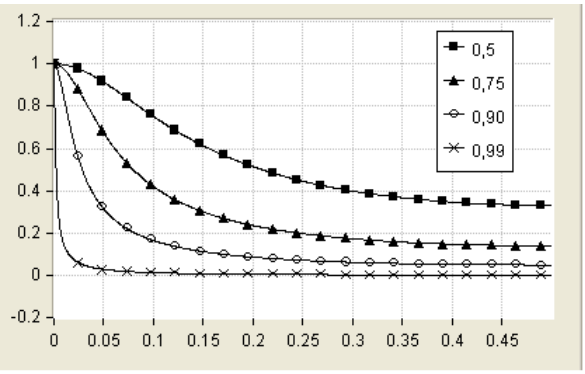
Pol prenosne funkcije je pri $z = \alpha$, iz tega sledi, da mora za stabilno prenosno funkcijo biti $|\alpha| \leq 1$, ustrezna diferenčna enačba je:

$$y_k = (1-\alpha)x_k + \alpha y_{k-1}$$

Amplitudna karakteristika tega sistema je:

$$\left|H(z = e^{i\omega T_V})\right| = \frac{1-\alpha}{|1-\alpha e^{-i\omega T_V}|} = \frac{1-\alpha}{\sqrt{1+\alpha^2 - 2\alpha\cos 2\pi\frac{\omega}{\omega_V}}}$$

Na sliki 13.4 je amplitudna karakteristika filtra z eksponentnim pozabljanjem (enostavnega nizkoprepustnega filtra) za različne vrednosti koeficienta α.



Slika 13.4: Amplitudna karakteristika filtra z eksponentnim pozabljanjem za različne koeficiente α, enota abscisne osi je ω_V

13.6. Filtriranje IIR

Za uspešno filtriranje potrebujemo vrednosti koeficientov, ki jih uporabimo v diferenčni enačbi. Te lahko izberemo na spodaj opisane načine. Pri tem prvi način temelji na izkušnjah in intuiciji, preostali trije pa na poznavanju prenosnih funkcij *T(s)* zveznih sistemov oziroma analognih filtrov, iz katerih je preko transformacij mogoče izračunati prave koeficiente. Postopki ne dajo natanko enakih koeficientov tudi pri enakih izhodiščnih enačbah, zato so tudi rezultirajoče frekvenčne karakteristike boljši ali slabši približki frekvenčnih karakteristik zveznih sistemov.

13.6.1. Polaganje ničel in polov v z ravnini

Frekvenčno karakteristiko sistema določa položaj ničel z_n in polov z_m v ravnini z. Pri tem je amplitudna karakteristika dana s kvocientom produkta razdalj med točko na enotskem krogu in položaji ničel ter produkta razdalj med isto točko na enotskem

krogu in položajem polov v ravnini *z*. Števec in imenovalec prenosne funkcije namreč lahko razbijemo na produkte, absolutna vrednost tako zapisane prenosne funkcije pa je enaka kvocientu produktov absolutnih vrednosti posameznih členov v imenovalcu in številu.

$$|H(z)| = \left|\frac{Y(z)}{X(z)}\right| = \left|\frac{a_N z^{-N} + \cdots + a_0 z^0}{b_M z^{-M} + \cdots + b_0 z^0}\right| = \left|\frac{\prod_{n=0}^{N}(z - z_n)}{\prod_{m=0}^{M}(z - z_m)}\right| = \frac{\prod_{n=0}^{N}|z - z_n|}{\prod_{m=0}^{M}|z - z_m|}$$

Z nekaj vaje je mogoče v ravnini *z* položiti ničle in pole tako, da se doseže želena amplitudna karakteristika. Pri tem si pomagamo z računalniškimi programi, ki sproti med postavljanjem in premikanjem ničel in polov računajo zgornji izraz in ustrezno frekvenčno karakteristiko. Lego ničel in polov pa nato uporabimo za formiranje *H(z)* in dalje diferenčnih enačb.

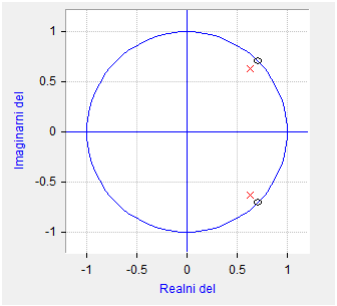
Zgled: filter za izločanje signala izbrane frekvence

Odziv filtra na vhodni signal je enak nič takrat, ko je vrednost kateregakoli člena v imenovalcu zgornje formule enaka nič. Pri določanju amplitude odziva spremenljivko z zamenjamo z $e^{i\omega T_v}$. Ko se povečuje frekvenca ω se torej gibljemo v nasprotni smeri urinega kazalca po krožnici *R*=1 od točke {1,0} v ravnini *z* do točke *z*{-1,0} pri frekvenci $\omega_v/_2$. Ko torej postavimo konjugirano kompleksni ničli na enotski krog pri kotu φ na primer $\pm\pi/_4$, je odziv filtra nič pri frekvenci $\omega_v/_8$. Konjugirano-kompleksni ničli zagotavljata realne koeficiente diferenčne enačbe, ki jo izpišemo kot:

$$y_k = x_k - 2x_{k-1}\cos\frac{\pi}{4} + x_{k-2}$$

Tak filter izloča željeno frekvenco, vendar je amplituda odziva v bližini izbrane frekvence nesimetrična, pa tudi pas oslabljenih signalov je relativno širok. Amplitudno karakteristiko filtra izostrimo tako, da v *z* ravnino dodamo par konjugirano kompleksnih polov, slika 13.5. Pola postavimo pod istim kotom kot ničle, pomaknemo pa ju malo v notranjost enotskega kroga. Naj bo razdalja polov od izhodišča podana z *R*, potem velja izkustvena formula:

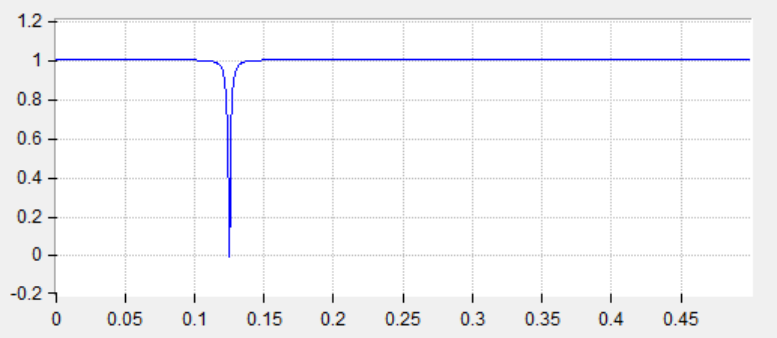
$$R = 1 - \pi\frac{BW}{f_v} \qquad \text{oziroma} \qquad BW = f_v\frac{1-R}{\pi}$$



Slika 13.5: Položaj ničel (krogec) in polov (križec) v z-ravnini za izločilni filter, R=0,9 (notch filter)

Pri tem parameter BW predstavlja širino zapornega pasu, če je $0{,}9 < R < 1$. Ustrezna diferenčna formula je:

$$y_k = x_k - 2x_{k-1}\cos\frac{\pi}{4} + x_{k-2} + 2Ry_{k-1}\cos\frac{\pi}{4} - R^2 y_{k-2}$$



Slika 13.6: Amplituda odzivav odvisnosti od frekvence za izločilni notch() filter, R=0,99, $\varphi = {}^{\pi}/_{4}$, vertikalna os predstavlja relativno amplitudo odziva, horizontalna pa f_v

13.6.2. Impulzno invariantna matoda

Temelji na poznavanju prenosne funkcije *T(s)* za analogni filter. Iz te prenosne funkcije s pomočjo obratne Laplace-ove transformacije izračunamo impulzni odziv *h(t)* filtra in ga povzorčimo v enakomernih časovnih intervalih T_v. Tako dobimo impulzni odziv $h(nT_V)$, ki ustreza danemu analognemu filtru. Prenosno funkcijo *H(z)* dobimo s pomočjo transformacije z, iz prenosne funkcije *H(z)* pa na koncu še izluščimo diferenčno enačbo digitalnega filtra (za podrobnosti glej ###).

Zgled: Določiti je treba diferenčno enačbo za nizkoprepustni filter drugega reda, ki ima optimalno obliko amplitudne karakteristike v bližini prelomne frekvence. Prelomna frekvenca $f_M$filtra naj bo 80Hz ($\omega_M = 502{,}65Hz$), frekvenca vzorčenja f_v pa znaša 1000Hz. Prototipna prenosna funkcija 2. reda za nizkoprepustni filter se glasi:

$$T(s) = \frac{1}{s^2 + \sqrt{2}s + 1}$$

Najprej priredimo prototipno prenosno funkcijo zahtevanim frekvencam. To dosežemo tako, da zamenjamo $s \to {}^{s}/_{\omega_M}$, saj gre za enostaven frekvenčni premik. Če bi bilo potrebno spreminjati tip filtra v visokopasovno, pasovno prepustno ali pasovno zaporno verzijo, bi uporabili zamenjave iz tabele 13.10.

$$T'(s) = T(s)|s \to s/\omega_M = \frac{\omega_M^2}{s^2 + \sqrt{2}\omega_M s + \omega_M^2} = \frac{C_1}{s - p_1} + \frac{C_2}{s - p_2}$$

Zgornje rešimo na: $p_1 = \frac{-\sqrt{2}\omega_M(1-i)}{2} = -355{,}427(1-i), \qquad p_2 = p_1^*$

In: $C_1 = \frac{-i\omega_M}{\sqrt{2}} = -355{,}427i, \qquad C_2 = C_1^*$

Prenosno funkcijo smo razstavili na člene, ki so prenosne funkcije prvega reda. Impulzni odziv h(t) sistema s prenosno funkcijo prvega reda je eksponentne oblike (Elektronika I).

$$H(s) = \frac{C}{s-p} \qquad \Rightarrow \qquad h(t) = C \cdot e^{pt} \qquad ; t \geq 0$$

Kot je navedeno v uvodu, se mora impulzni odziv digitalne verzije filtra ujemati z impulznim odzivom analognega prototipa pri časih kT_v, torej:

$$h(kT_v) = h(t)|(t = kT_v) = C \cdot e^{pkT_v}$$

Potrebujemo z-transformiranko impulznega odziva, zato zapišemo:

$$H(z) = \sum_{k=0}^{\infty} h(kT_v)z^{-k} = \sum_{k=0}^{\infty} C \cdot e^{pkT_v}z^{-k} = \frac{C}{1 - e^{pT_v}z^{-1}}$$

Torej velja:

$$H(s) \;\rightarrow\; H(z) \qquad \equiv \qquad \frac{C}{s-p} \;\rightarrow\; \frac{C}{1-e^{pT_v}z^{-1}}$$

Kadar imamo opravka z zvezno prenosno funkcijo, ki je sestavljena iz več funkcij prvega reda, je z-transformiranka impulznega odziva podana kot vsota odzivov na vsako posamezno funkcijo.

$$\sum_m \frac{C_m}{s-p_m} \quad \rightarrow \quad \sum_m \frac{C_m}{1-e^{p_m T_v}z^{-1}}$$

Kadar imamo opraviti s filtrom drugega reda, kot v tej nalogi, se zgornja transformacija poenostavi v:

$$\frac{C_1}{s-p_1} + \frac{C_2}{s-p_2} \quad \rightarrow \quad \frac{C_1}{1-e^{p_1 T_v}z^{-1}} + \frac{C_2}{1-e^{p_2 T_v}z^{-1}}$$

Ker sta pola p_1 in p_2 konjugirano kompleksna, prav tako sta konjugirano kompleksna tudi faktorja C_1 in C_2, se desni zgornji izraz naprej poenostavi v:

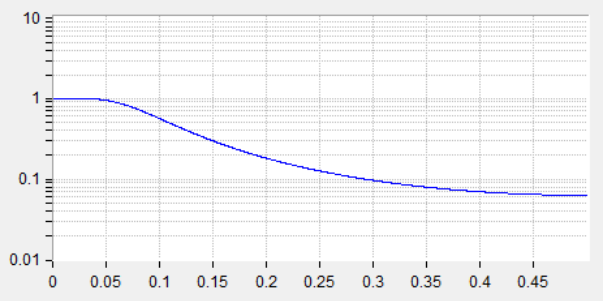
$$\frac{2C_r - [C_r\cos(p_i T_v) + C_i \sin(p_i T_v)]2e^{p_r T_v}z^{-1}}{1 - 2e^{p_r T_v}\cos(p_i T_v)\, z^{-1} + e^{2p_r T_v}z^{-2}}$$

Pri tem predstavlja C_r realni del koeficienta C, C_i pa imaginarni del. Podobno velja za p_r in p_i (realni del in imaginarni del za pol p). Če v zgornjo formulo umestimo prej izračunane vrednosti za p in C, dobimo:

$$H(z) = \frac{173{,}375 \cdot z^{-1}}{1 - 1{,}31436 \cdot z^{-1} + 0{,}491225 \cdot z^{-2}}$$

Od tod izpišemo rekurzivno formulo:

$$y_k = 173{,}375 \cdot x_{k-1} + 1{,}31436 \cdot y_{k-1} - 0{,}491225 \cdot y_{k-2}$$



Slika 13.7: Normirana karakteristika filtra, koeficienti so izračunani s pomočjo impulzno-invariantne metode; ordinatna os predstavlja ojačanje, abscisna pa f/f_v. Ojačanje pade na -3dB (0,707) pri frekvenci 80Hz.

13.6.3. Metoda enakega z-transforma

Tudi tu je potrebno poznati prenosno funkcijo analognega filtra. Iz prenosne funkcije izračunamo lego ničel in polov v ravnini s za analogni filter, nato pa lego prenesemo v ravnino *z* s pomočjo izraza:

$$(s - a) \rightarrow (1 - z^{-1}e^{aT_v})$$

Lego prenešenih ničel in polov uporabimo za konstrukcijo prenosne funkcije, iz katere izpišemo diferenčno enačbo.

Zgled: Določiti je treba diferenčno enačbo za nizkoprepustni filter drugega reda, ki ima optimalno obliko amplitudne karakteristike v bližini prelomne frekvence. Prelomna frekvenca $f_M$filtra naj bo 80Hz ($\omega_M = 502{,}65Hz$), frekvenca vzorčenja f_v pa znaša 1000Hz. Prototipna prenosna funkcija 2. reda za nizkoprepustni filter se glasi:

$$T(s) = \frac{1}{s^2 + \sqrt{2}s + 1}$$

Najprej priredimo prototipno prenosno funkcijo zahtevanim frekvencam. To dosežemo tako, da zamenjamo $s \rightarrow {}^{s}/_{\omega_M}$, saj gre za enostaven frekvenčni premik. Če bi bilo potrebno spreminjati tip filtra v visokopasovno, pasovno prepustno ali pasovno zaporno verzijo, bi uporabili zamenjave iz tabele 13.10.

$$T'(s) = T(s)|s \rightarrow s/\omega_M = \frac{\omega_M^2}{s^2 + \sqrt{2}\omega_M s + \omega_M^2}$$

Iz zapisane prenosne funkcije izračunamo lego polov:

$$p_{1,2} = \frac{\sqrt{2}\omega_M}{2}(-1 - i)$$

Kar razdelimo v realni in imaginarni del:

$$p_r = -355{,}427 \qquad p_i = 355{,}427i$$

V splošnem je prenosna funkcija analognega filtra podana kot kvocient polinomov, lahko pa jo zapišemo tudi faktorizirano v obliki:

$$T(s) = \frac{(s-z_1)(s-z_2)\cdots(s-z_M)}{(s-p_1)(s-p_2)\cdots(s-p_N)} \quad , z_k \; in \; p_k \; predstavljajo \; ničle \; in \; pole$$

V začetku poglavja navedeno transformacijo lahko uporabimo na vsakem faktorju posebej. Kadar imamo opravka s filtrom drugega reda, se prenosna funkcija reducira v:

$$T(s) = \frac{(s-z_1)(s-z_2)}{(s-p_1)(s-p_2)}$$

Ko izvedemo transformacijo na tej prenosni funkciji, dobimo:

$$\frac{(s-z_1)(s-z_2)}{(s-p_1)(s-p_2)} \rightarrow \frac{1-(e^{z_1T_v}+e^{z_2T_v})z^{-1}+e^{(z_1+z_2)T_v}z^{-2}}{1-(e^{p_1T_v}+e^{p_2T_v})z^{-1}+e^{(p_1+p_2)T_v}z^{-2}}$$

Ker nastopajo ničle in poli v konjugirano kompleksnih parih, se transformacija reducira v:

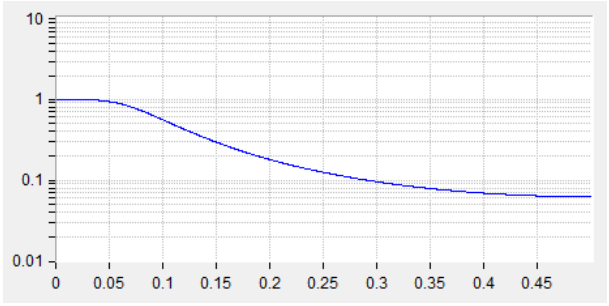
$$\frac{1-2e^{z_rT_v}\cos z_iT_v\, z^{-1}+e^{z_rT_v}z^{-2}}{1-2e^{p_rT_v}\cos p_iT_v\, z^{-1}+e^{2p_rT_v}z^{-2}}$$

Ko v ta izraz vstavimo realni in imaginarni del vrednosti pola in upoštevamo, da prenosna funkcija v zastavljeni nalogi nima ničel, dobimo diskratno verzijo prenosne funkcije:

$$H(z) = \frac{252657{,}0225}{1-1{,}314135z^{-1}+0{,}4912243z^{-2}}$$

Od tod izpišemo diferenčno enačbo:

$$y_k = 252657{,}0225 \cdot x_k + 1{,}314249 \cdot y_{k-1} - 0{,}70091826 \cdot y_{k-2}$$



Slika 13.8: Normirana karakteristika filtra, koeficienti so izračunani s pomočjo metode enakega z-transforma; ordinatna os predstavlja normirano ojačanje, abscisna pa f/f_v

13.6.4. Bilinearna transformacija

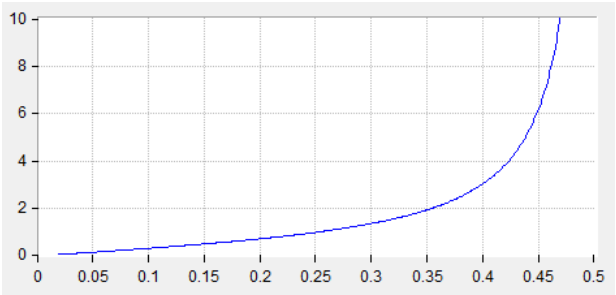
Je najpogosteje uporabljana metoda za računanje koeficientov diferenčne enačbe. Poznati moramo prenosno funkcijo *T(s)* analognega filtra, ki ustreza zahtevam za filtriranje signala. Prenosno funkcijo *T(s)* s pomočjo bilinearne transformacije spremenimo v prenosno funkcijo *H(z)* digitalnega filtra, iz katere izpišemo diferenčno enačbo.

Bilinearno transformacijo opravimo z zamenjavo:

$$s \leftrightarrow 2f_v \frac{1-z^{-1}}{1+z^{-1}} \quad \text{, pri tem je} \quad f_v = \frac{1}{T_V}$$

Pri tej vrsti transformacije pride do popačitve, zaradi katere frekvenčni odziv digitalnega filtra ni enak odzivu analognega prototipa. Popačitev frekvenčne skale ovrednotimo tako, da v zgornjem pravilu za zamenjavo nadomestimo $s \to i\omega_a$ in $z \to e^{i\omega_d T_V}$ ter izračunamo relacijo med ω_a in ω_d. Ta znaša:

$$\omega_a = 2f_v \cdot \tan\frac{\pi \cdot \omega_d}{\omega_v}$$



Slika 13.9: Preslikava frekvenc med z bilinearno transformacijo dobljenim filtrom in prototipom analognega filtra; abscica je f_d/f_v, ordinata pa f_a

Prelomne frekvence digitalnega filtra so torej premaknjene glede na prelomne frekvence analognega filtra, premik podaja zgornja enačba. Premik je tem večji, čim bliže je prelomna frekvenca Nyquist-ovi meji. Ker je premik znan, ga lahko v naprej kompenziramo z nasprotnim premikom. Če želimo prelomno frekvenco digitalnega filtra pri ω_d, moramo začeti načrtovanje z analognim filtrom z mejno frekvenco ω_a.

Zgled:

Določiti je treba diferenčno enačbo za nizkoprepustni filter drugega reda, ki ima optimalno obliko amplitudne karakteristike v bližini prelomne frekvence. Prelomna frekvenca $f_M$filtra naj bo 80Hz ($\omega_M = 502{,}65Hz$), frekvenca vzorčenja f_v pa znaša 1000Hz. Prototipna prenosna funkcija 2. reda za nizkoprepustni filter se glasi:

$$T(s) = \frac{1}{s^2 + \sqrt{2}s + 1}$$

a) Najprej kompenziramo premik prelomne frekvence, ki ga povzroči bilinearna transformacija. Nova prelomna frekvenca ω'_M znaša:

$$\omega'_M = 2f_v \cdot \tan\frac{\pi \cdot f_M}{f_v} = 2 \cdot 1000 \cdot \tan\frac{\pi \cdot 80}{1000} = 513{,}51Hz$$

To je malo več, kot znaša osnovna prelomna frekvenca, kar je pravilno, saj bo po bilinearni transformaciji frekvenčna skala stisnjena.

b) Nato modificiramo prototipno prenosno funkcijo $T(s)$ tako, da odraža dejansko vrsto filtra in kompenzirano prelomno frekvenco. Spodnja tabela 13.10 podaja potrebne zamenjave:

Tip filtra	Potrebna zamenjava
$LP \rightarrow LP$	$s \rightarrow \frac{s}{\omega'_M}$
$LP \rightarrow HP$	$s \rightarrow \frac{\omega'_M}{s}$
$LP \rightarrow BP$	$s \rightarrow \frac{s^2 + \omega_0^2}{Ws}$
$LP \rightarrow BS$	$s \rightarrow \frac{Ws}{s^2 + \omega_0^2}$
Definicija: $W = \omega'_{M2} - \omega'_{M1}$ in $\omega_0^2 = \omega'_{M1} \cdot \omega'_{M2}$	

V našem primeru torej zapišemo:

$$T'(s) = T(s)\big|_{s \rightarrow \frac{s}{\omega'_M}} = \frac{\omega'^2_M}{s^2 + \sqrt{2}s\omega'_M + \omega'^2_M} = \frac{263692{,}52}{s^2 + 726{,}21s + 263692{,}52}$$

c) Sledi postopek bilinearne transformacije:

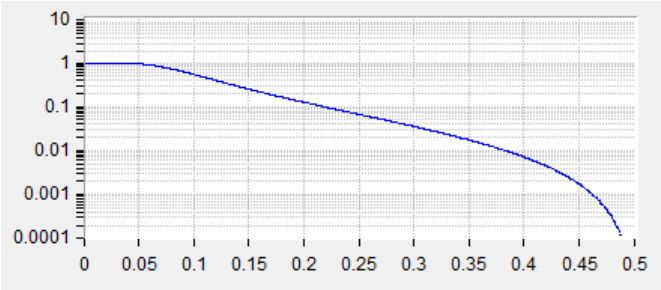
$$H(z) = T'(s)\big|_{s \rightarrow 2f_v \cdot \frac{1-z^{-1}}{1+z^{-1}}}$$

$$H(z) = \frac{263692{,}52}{4f_v^2\left(\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + 726{,}21 \cdot 2 \cdot f_v \cdot \frac{1-z^{-1}}{1+z^{-1}} + 263692{,}52}$$

$$H(z) = \frac{0{,}0461314 \cdot (1 + 2z^{-1} + z^{-2})}{1 - 1{,}3072884z^{-1} + 0{,}491814z^{-2}} = \frac{Y(z)}{X(z)}$$

d) Iz prenosne funkcije izpišemo diferenčno enačbo:

$$y_k = 0{,}0461314 \cdot (x_k + 2 \cdot x_{k-1} + x_{k-2}) + 1{,}3072884 \cdot y_{k-1} - 0{,}491814 \cdot y_{k-2}$$

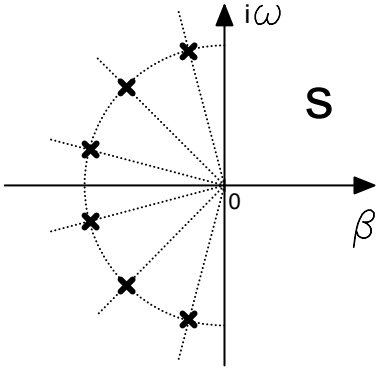


Slika 13.11: Amplitudna karakteristika za zgornjo diferenčno enačbo, vertikalna os predstavlja ojačanje, horizontalna pa frekvenco do $f_v/2$. Dobro se vidi vpliv stisnenja frekvenčne skale pri frekvencah blizu $f_v/2$, ko ojačanje hitreje pada.

13.7. IIR filtri višjega reda

Enostavni filter, kot je bil predstavljen v zgledu v prejšnjem poglavju, le slabo loči signale željenih frekvenc of tistih, ki jih ne želimo. Za boljše ločevanje potrebujemo filtre višjih redov, take popisuje prenosna funkcija *H(z)* z več poli in ničlami. Do take pridemo, če začnemo s prenosno funkcijo *T(s)* višjega reda, ki ima v ravnini *s* več polov. Po Butterworthu dobimo najbolj ravno amplitudno karakteristiko v prepustnem delu in hkrati najboljše definiran prehod med prepustnim in zapornim delom amplitudne karakteristike takrat, ko pole prenosne funkcije *T(s)* razporedimo ekvidistančno po krožnici v ravnini *sI,* leva polovica, slika 13.12.

Iz znanih položajev polov v ravnini *s* napišemo prenosno funkcijo *T(s)* ter jo predelamo s pomočjo bilinearne transformacije (ali katerega drugega postopka) v ustrezno *H(z)*. Postopek je standardiziran in ga navadno opravimo s pomočjo računalniškega programa.

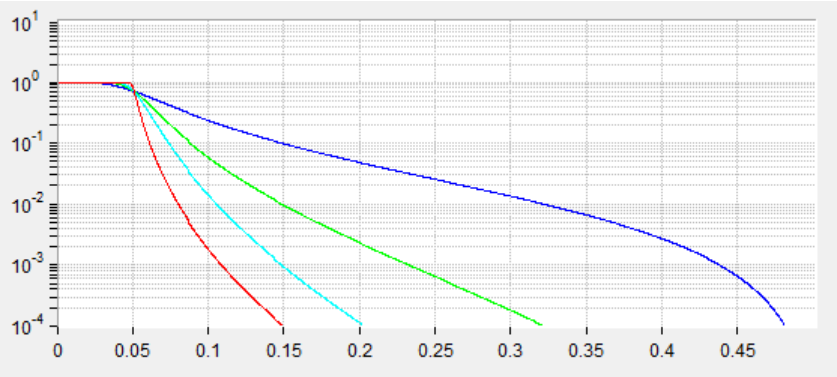


Slika 13.12: Poli za prenosno funkcijo šestega reda po Butterworthu v ravnini s; vsi poli so levo od imaginarne osi, kar zagotavlja stabilnost filtra, razporejeni so ekvidistančno

Če lahko toleriramo valovitost v prepustnem delu karakteristike, s premikanjem polov v ravnini *s* dosežemo še boljše ločevanje prepustnega dela karakteristike od zapornega dela. Takrat pole prerazporedimo s krožnice na elipso

po Čebiševu. Elipsa je vertikalno orientirana, njeno središče je v izhodišču ravnine *s*. Sploščenost elipse definira valovitost v prepustnem pasu. Bolj, ko je elipsa sploščena, večja je valovitost. Navadno toleriramo valovitost do največ nekaj odstotkov.

Večje število polov v prenosni funkciji povzroči, da so koeficienti diferenčne enačbe po velikosti močno različni. Ko razlike dosežejo nekaj velikostnih redov, lahko to v računalniku pri računanju s celimi števili privede do numeričnih težav, ki pokvarijo karakteristiko filtra. Takrat filter raje sestavimo tako, da zaporedno vežemo več enostavnih filtrov drugega reda, v vsaki od stopenj pa implementiramo en konjugirano kompleksen par polov od množice polov v ravnini *s*.



Slika 11.40: Primerjava amplitudnih karakteristik za filter drugega (modro), četrtega (zeleno) in šestega (cian) reda po Butterworthu ter šestega (rdeče) reda po Čebiševu (valovitost 3%).

Kadar želimo povečati dušenje v zapornem delu karakteristike, dodamo na imaginarno os pare konjugirano kompleksnih ničel nad točko, kjer se navidezna krožnica po Butterworthu ali elipsa po Čebiševu dotika ordinatne osi. Frekvenčno karakteristiko ovrednotimo tako, da *s* v *T(s)* zamenjamo z $i\omega$ in določimo absolutno vrednost tako dobljenega izraza. To pomeni, da se med analizo točka interesa pomika po ordinatni osi od izhodišča navzgor. Kadar točka interesa povozi ničlo na ordinatni osi, je amplituda odziva filtra enaka nič, torej je takrat dušenje neskončno veliko. S primerno razporeditvijo ničel lahko »pripnemo« amplitudo odziva na vrednost nič pri frekvencah, ki so za nas interesentne.

14. Generiranje signalov

14.1. Generiranje signalov – sprotno računanje in tabele

Signale najlaže generiramo tako, da uporabimo predpripravljene matematične funkcije, ki so na razpolago v vsakem programskem jeziku. Tako lahko v zaporednih obhodih skozi zanko kličemo matematično funkcijo, vsakokrat z naslednjim argumentom.

Zgled: Generiraj harmonski signal s periodo 32 ponovitev zanke.

Spodnji meta-program spremenljivki Y priredi ob vsakem obhodu skozi zanko novo vrednost. Za periodičnost argumenta n% poskrbi funkcija AND.

```
Do
   Y = Ampl * Sin (2 * pi * n% / 32)
   n% = (n% + 1) And 31
Loop While (True)
```

Tako generiranje je univerzalno uporabno, kadar imamo na razpolago dovolj močan računalnik ali/in dovolj časa. Program navadno računa matematične funkcije s pomočjo vrste, kar pa zna biti časovno potratno in lahko onemogoči sprotno računanje. Takrat se lahko zatečemo k dejstvu, da imamo opraviti s periodičnim signalom, zato lahko njegove vrednosti enkrat v naprej izračunamo in shranimo v polje, v zanki pa kasneje le beremo elemente polja.

Zgled: Enako, kot zgoraj.

```
Dim a(32) as Single                          ' definiraj polje
For n% = 0 to 31
   a(n%) = Ampl * Sin (2 * pi * n% / 32)     ' napolni polje
Next n%
Do
   Y = a(n%)                                 ' uporabi polje
   n% = (n% + 1) And 31                      ' kazalec v polje
Loop While (True)
```

Če generiramo signale sproti v realnem času, lahko vedno izberemo primerno periodo izvajanja zanke, morda s pomočjo prekinitev, ki jih povzroča urin signal. Logična funkcija AND in inkrementiranje v zanki navadno ne predstavlja bistvene

potrate časa. Žal lahko na ta način generiramo le signale, katerih perioda je enaka celoštevilčnemu mnogokratniku časa, potrebnega za en obhod skozi zanko. Kadar je potrebno generirati posamičen signal, omejitev ne predstavlja problema, saj je treba le časovni interval prekinitev prirediti zahtevani frekvenci signala in številu vzorcev v periodi. Zatakne se takrat, ko je treba generirati več signalov z različnimi frekvencami hkrati. Zatakne se tudi takrat, ko uporabljeni sistem nima dovolj prostora za shranjevanje polja ali takrat, ko nismo pripravljeni čakati na začetku do izračuna vrednosti elementov polja.

14.2. Hitro generiranje harmonskih signalov

Pogosto potrebujemo harmonski signal, a je njegovo sprotno računanje v realnem času preko trigonometričnih funkcij preveč zamudno. Lahko si pomagamo s tabelo, v katero v naprej zapišemo potrebne vrednosti trigonometrične funkcije, v realnem času pa iz te tabele le beremo v naprej pripravljena števila. Tako način generiranja je preprost in zanesljiv, vendar deluje le za eno frekvenco, pa še te med generiranjem ne moremo spreminjati.

Kadar potrebujemo signal drugačne ali celo spremenljive frekvence, postopamo drugače. Definirajmo kompleksni signal $y_c = Ae^{i\omega_h t}$, njegova frekvenca znaša ω_h, amplituda pa je *A*. Ko tak signal diskretiziramo, izračunamo njegovo trenutno vrednost y_{ck} s pomočjo njegove prejšnje vrednosti in koeficienta $e^{i\omega_h T_v} = e^{i2\pi\, f_h/f_v} = e^{i\varphi}$ po formuli:

$$y_{c\,k} = y_{c\,k-1} \cdot e^{i\varphi}$$

V resničnem svetu kompleksni signal popišemo z dvema signaloma, prvim za realni del (y_{Ik}) in drugim za imaginarni del (y_{Qk}).

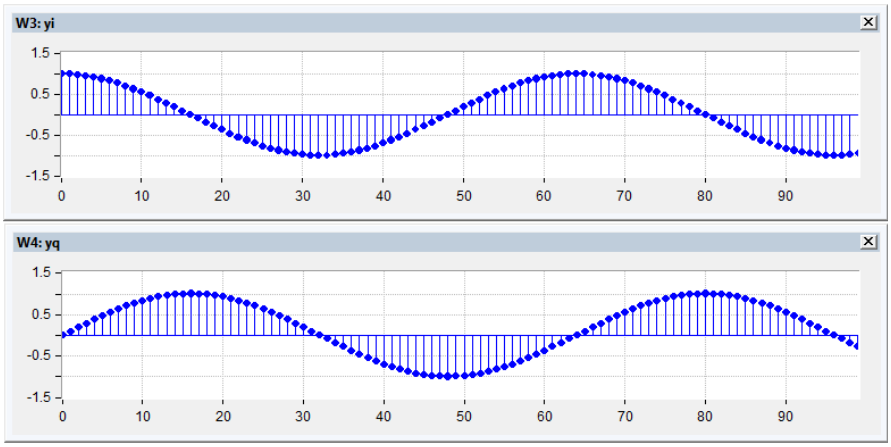
$$y_{Ik} + iy_{Qk} = \left(y_{I\,k-1} + iy_{Q\,k-1}\right) \cdot (\cos\varphi + i\sin\varphi)$$

Od tod izpišemo izraza za dva sklopljena signala:

$$y_{Ik} = y_{I\,k-1} \cdot \cos\varphi - y_{Q\,k-1} \cdot \sin\varphi$$
$$y_{Qk} = y_{I\,k-1} \cdot \sin\varphi + y_{Q\,k-1} \cdot \cos\varphi$$

Za začetek postavimo vrednosti $y_{I0} = 1$ in $y_{Q0} = 0$ ter nadaljujemo z računanjem po zgornih formulah. Na sliki 14.1 sta narisana signala y_i in y_q. Nekaj previdnosti je potrebno, če računamo celoštevilčno ali s števili z enojnim zapisom, saj napake pri zaokroževanju lahko povzročijo počasno spreminjanje amplitude signalov. Takrat je smiselno uvesti regulacijsko zanko, ki ohranja amplitudo signala, ali pa povečati število bitov v zapisu trenutne vrednosti signala.

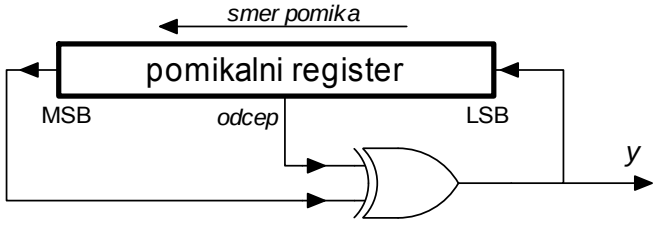
Tako generiran signal ima dodatno lepo lastnost, na razpolago sta namreč osnovni signal in njegova zo 90 stopinj zakasnjena verzija, ki je zato ni treba posebej generirati na primer s Hilbertovo transformacijo.



Slika 14.1: Generirana harmonska signala

14.3. Generiranje naključnih signalov

Tudi naključne signale lahko generiramo z uporabo funkcij, ki so vgrajene v višje programske jezike. Kadar ustraznih funkcij ni na razpolago ali pa uporabljeni sistem ne dopušča sprotnega računanja naključne funkcije iz programskega jezika, lahko uporabimo takoimenovani generator navidezno naključnega zaporedja bitov (»pseudo random bit sequence«). Sestavlja ga pomični register dolžine M bitov in vrata XOR po shemi na sliki 14.2.



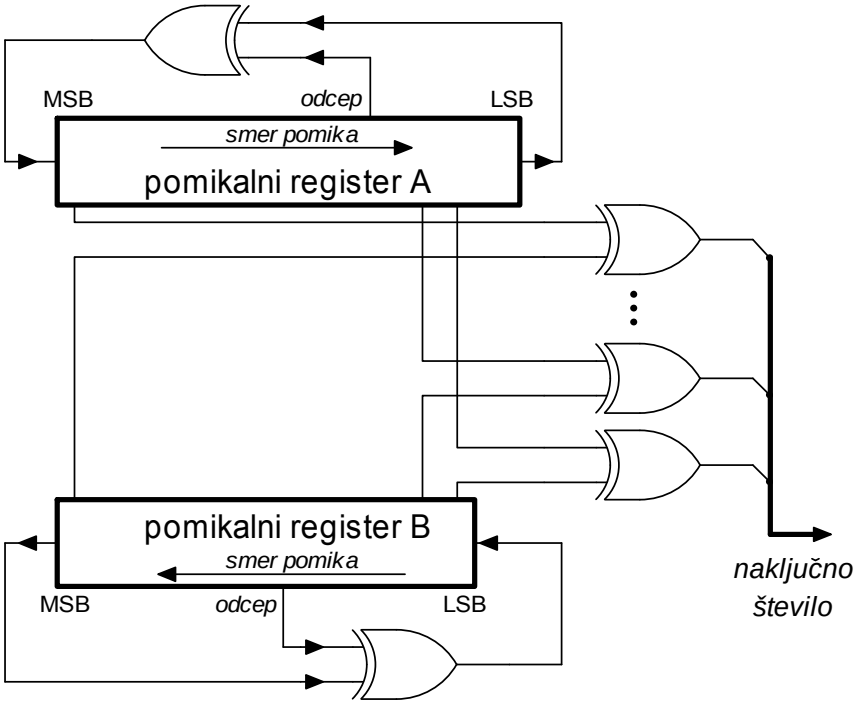
Slika 14.2: Generator na videz naključnega zaporedja bitov in signal na njegovem izhodu

Tak generator daje navidez naključno zaporedje bitov na izhodu *y*, perioda je odvisna od števila bitov v pomičnem registru. Za najdaljšo možno periodo moramo uporabiti prave odcepe iz pomičnega registra, kot so definirani na primer v literaturi (»Numerical recepies...«, tam sta opisani dve verziji za implementacijo v strojni opremi in v programski opremi).

Če potrebujemo naključno število namesto naključnega bita, bi najprej pomislili na celotno vsebino pomičnega registra, ne le na najmanj pomemben bit le-tega. Žal

celovna vsebina ni posebej naključna, saj je vsaka naslednja vrednost dvakratnik prejšnje, morda povečan za vrednost LSB. Treba bo torej poskusiti kaj drugega.

Uporabimo lahko dva generatorja naključnega zaporedja bitov po sliki 14.3. Vsako od števil v posamičnem generatorju ni dovolj dobro, če pa ju združimo s pomočjo funkcije niza vrat XOR, je kakovost tako dobljenega števila bistveno boljša. Pri tem v generatorjih uporabimo pomična registra različnih dolžin, zato je skupna perioda generiranega števila enaka produktu period naključno generiranih bitov posamičnega registra. Poskrbimo še, da registra pomikata svojo vsebino v nasprotnih smereh, kar dodatno premeša generirana števila.



Slika 14.3: Tako lahko generiramo naključno število z enakomerno porazdelitvijo

Števila, ki jih dobimo iz takega generatorja, so enakomerno porazdeljena po območju, ki ga definira število bitov v obeh pomičnih registrih in število vrat XOR.

Kadar tak generator naključnih števil implementiramo v programski opremi računalnika, izgleda to takole (»prbsA« in »prbsB« sta 32-bitna pomična registra, »rnd« je izračunano naključno število, ki ga sestavlja 12 bitov, »^« predstavlja operacijo XOR, »&« pa operacijo AND):

```
rnd = 0;
if (prbsB & 0x400000) {            // če je MSB == 1 potem
  prbsB = prbsB ^ 0x10;            // odcep pri četrtem bitu
```

```
      prbsB = (prbsB << 1) + 1; }
    else
      prbsB = (prbsB << 1);

    if (prbsA & 0x1) {                  // če je LSB == 1 potem
      prbsA = prbsA ^ 0x200000;         // odcep pri 22. bitu
      prbsA = (prbsA >> 1) + 0x200000; }
    else
      prbsA = (prbsA >> 1);

    rnd = (prbs1 ^ prbs2) & 0xfff;
```

Kadar potrebujemo naključna števila z Gausovo porazdelitvijo, lahko povprečimo večje število naključnih števil z enakomerno porazdelitvijo. Literatura ### priporoča, da povprečimo več kot deset naključnih števil.

Zgled: Generator naključnih števil z enakomerno porazdelitvijo, če imamo na razpolago funkcijo Rnd, ki daje naključna števila. Zadnja vrstica programa poskrbi, da je povprečna vrednost signala enaka 0 (odšteje enosmerno komponento) in prilagodi njegovo velikost.

```
Nsample = Rnd
For i = 1 To 9
   Nsample = Nsample + Rnd
Next i
Nsample = (Nsample / 10 - 0.5) * Amplitude
```

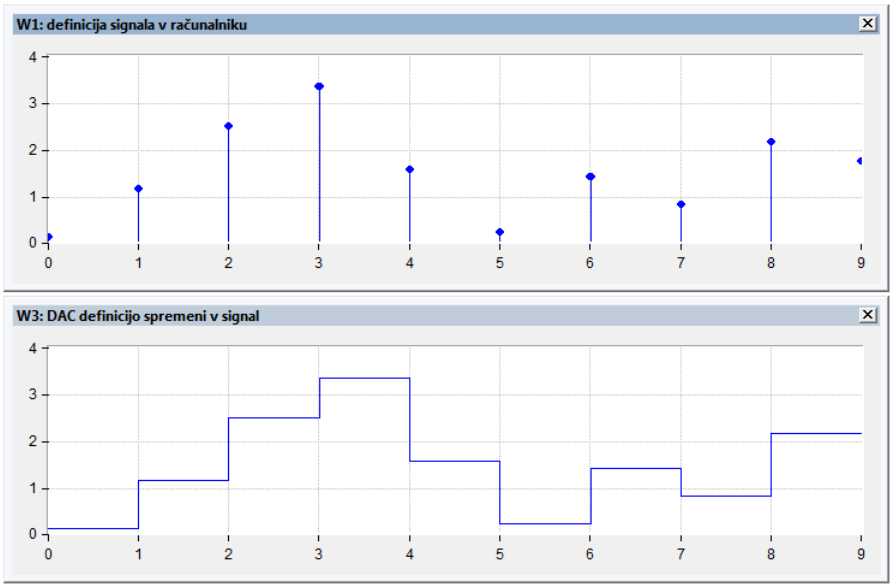
14.4. Korekcija frekvenčne karakteristike DAC

Signale, kot jih predstavljamo v računalniku, sestavljajo utežene delta funkcije (slika 11.2). Vrednost je znana le ob mnogokratnikih časa vzorčenja T_v. Ko take signale generiramo, navadno uporabimo digitalno-analogni poretvornik (DAC), ki zadrži generirano vrednost med mnogokratniki časa vzorčenja, slika 14.4. Opisana lastnost pretvornika vpliva na frekvenčno karakteristiko generiranega signala, ki ni več enaka frekvenčni karakteristiki signala v računalniku.

Zadrževalno lastnost DACa analiziramo modeliramo z odzivom po sliki 14.5. DAC zadrži vrednost za čas T_v do naslednjega sunka, podani odziv pa predstavlja *h(t)*, odziv na vzbujanje z delta sunkom.

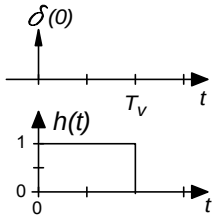
Velikost odziva v odvisnosti od frekvence signala digitalno-analognega pretvornika določimo kot absolutno vrednost Fourierovega transforma impulznega odziva:

$$\left|F\left(h(t)\right)\right| = \int_{-\infty}^{\infty} h(t)e^{-i\omega t}\,dt = T_v \frac{\sin\left(\pi \frac{f}{f_v}\right)}{\pi \frac{f}{f_v}}$$



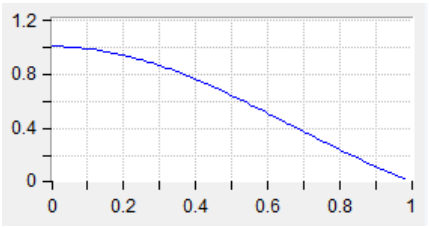
Slika 14.4: Zadrževalna lastnost DAC, abscisa je v T_v, ordinata v poljubnih enotah

Opraviti imamo z znano funkcijo sin(x)/x, njen graf je na sliki 14.6. Signali v sistemu imajo frekvenco največ $f_v/2$, tam ima funkcija vrednost 0,63 (-3,92dB) tiste, ki jo ima pri frekvenci nič. DAC torej podaja signale s frekvenco, ki je blizu Nyquistove meje z amplitudo, ki je manjša od amplitude za signale manjših frekvenc.



Slika 14.5: Odziv na delta sunek za DAC

Navedeno pomanjkljivost korigiramo z dodatnim filtrom, ki ga vgradimo v programsko opremo pred pošiljanjem podatkov na DAC. Ta filter poudarja signale visokih frekvenc v taki meri, da ravno kompenzira slabost DAC. Že z enostavnim IIR filtrom prvega reda, poglavje 13.1, lahko to napako zmanjšamo na 1dB. Prenosna funkcija filtra je podana z:



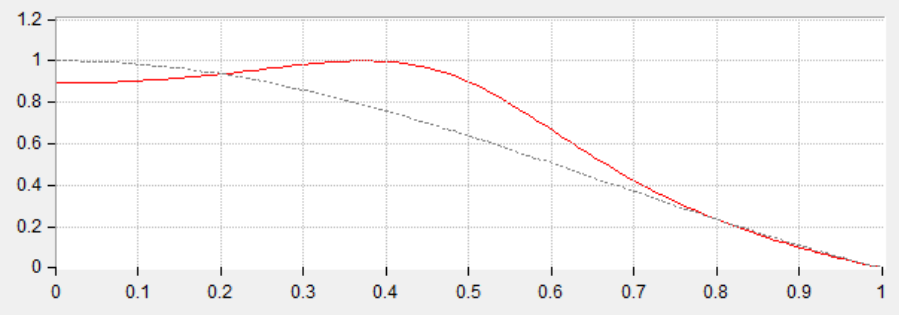
Slika 14.6: Amplituda odziva digitalno-analognega pretvornika, abscisa v f/f_v, ordinata je normirana

$$H(z) = \frac{b}{1 + az^{-1}}$$

Ustrezna amplitudna karakteristika znaša:

$$|H(z = e^{\omega T_v})| = \frac{b}{\sqrt{1 + a^2 + 2a\cos\left(2\pi\frac{f}{f_v}\right)}}$$

Za koeficienta $a = 0{,}224$ in $b = 1{,}092$ (####) dosežemo minimalno valovitost v pasu od 0 do Nyquistove frekvence, slika 14.7.



Slika 14.7: Kompenzirana amplituda odziva (rdeče), in originalna amplituda odziva (sivo), abscisa je v f/f_v, ordinata je normirana

15. Modulacije in demodulacije

15.1. Amplitudna modulacija in demodulacija

Pri amplitudni modulaciji signala se velikost nosilnega signala x_N spreminja sinhrono s trenutno vrednostjo modulacijskega signala x_M. Amplitudno modulirani signal X_{AM} zapišemo:

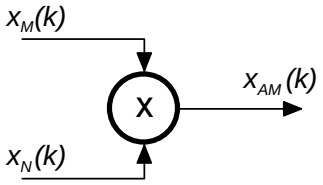
$$x_{AM} = x_M \cdot x_N = x_M \cdot \sin \omega_N t$$

Pri tem je ω_N frekvenca nosilnega signala. Kadar je modulacijski signal harmonske oblike, dobimo znani izraz:

$$x_{AM} = \sin(\omega_M t) \cdot \sin(\omega_N t) = \frac{1}{2}\left(\sin\left((\omega_N + \omega_M)t\right) + \sin\left((\omega_N - \omega_M)t\right)\right)$$

15.1.1. Amplitudna modulacija

Amplitudno modulirani signal je sestavljen in dveh komponent, ki imata frekvenci enaki vsoti in razliki obeh, v modulacijski shemi uporabljenih frekvenc. Najlaže ga izdelamo tako, da povzorčena ali digitalno generirana signala x_M in x_N množimo vzorec za vzorcem, produkt pa pošiljamo na digitalno-analogni pretvornik, slika 15.1.



Slika 15.1: Shema računanja amplitudno moduliranega signala

15.1.2. Sprotno določanje amplitude signala

Naš namen ni generirati amplitudno modulirani signal, pač pa iz danega amplitudno moduliranega signala izluščiti trenutno vrednost modulacijskega signala. Pri meritvah namreč pogosto vzbujamo merilni sistem s harmonskim signalom konstantne amplitude, odziv sistema pa predstavlja veličino, ki na sistem vpliva. Odziv ima prav tako harmonsko obliko in isto frekvenco, zanima nas le velikost.

Ker je velikost odziva sistema sorazmerna tudi z amplitudo vzbujanja, se zdi smiselno zajemati dva signala: signal za vzbujanje sistema x_N in odziv sistema x_{AM}. V postopku merjenja bomo iz obeh zajetih signalov izluščili njihovi amplitudi, nazadnje

pa določili njun kvocient in tako izmerili le vpliv merjene veličine odziv sistema, slika 15.2.

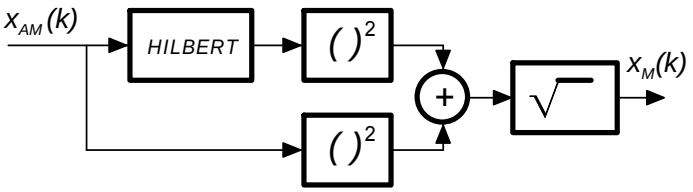
Od tu naprej bomo pogledali, kako iz enega od izmerjenih signalov izluščimo njegovo amplitudo, naj bo to x_{AM}. Za drugi signal bi postopali enako. Pred obdelavo s pasovno prepustnim filtrom iz zajetega signala odstranimo vse komponente, ki za nas niso zanimive.

Velikost signala x_{AM} lahko določimo tako, da izračunamo njegovo absolutno vrednost. Absolutna vrednost harmonskega signala da nihajoči signal z dvojno frekvenco ω_{AM}, iz katerega se da dvakrat v periodi zaznati maksimum, ki predstavlja amplitudo. Tako merjenje je dokaj nezanesljivo, saj vsak šum pokvari izmerjeno vrednost. Poleg tega je amplituda znana le dvakrat v periodi, kar morda ni dovolj.

Težavam se enostavno ognemo z uporabo Hilbertovega transforma po shemi 15.2. Zajeti signal po Hilbertovi transformaciji $H(x_{AM})$ ima enako amplitudo, kot jo ima originalni signal, le fazno je zakasnjen za 90 stopinj. Iz originalnega signala in njegove zakasnjene verzije sestavimo amplitudo x_M po znani formuli:

$$x_M = \sqrt{\left(H(x_{AM})\right)^2 + (x_{AM})^2}$$

Pri tem se zdi smiselno zajemati signal x_{AM} s frekvenco, ki je približno štirikratnik njegove frekvence. V tem primeru delamo na sredini Nyquistove področja in že majhno število koeficientov v jedru Hilbertovega transforma zadošča.



Slika 15.2: Shema za določanje amplitude signala z uporabo Hilbertove transformacije

15.1.3. Približek za hitro računanje kvadratnega korena vsote

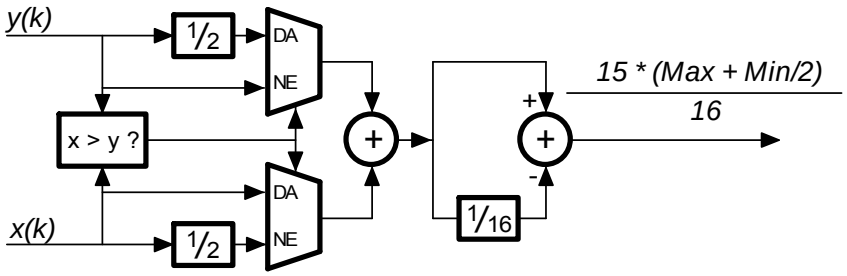
Računanje kvadratnega korena vsote dveh argumentov *x* in *y* je enostavna naloga za osebni računalnik pri majhnih frekvencah vzorčenja. Ta naloga postane pretežka pri velikih frekvencah vzorčenja ali šibkejših sredstvih za računanje. Takrat se zadovoljimo z aproksimacijo vrednosti korena, ki jo imanujejo »αMax+βMin«. Pri tej aproksimaciji moramo vedeti, kateri od argumentov je večji, tega označimo z *Max*, drugega z *Min*. Verjamemo, da je vrednost korena blizu vrednosti izraza:

$$x_M = \alpha \cdot Max + \beta \cdot Min$$

Računanje kvadratnega korena smo tako zamenjali z računanjem dveh produktov in ene vsote, kar je nedvomno hitrejše. Obstajajo tabele z vrednostmi za α(0,96043387) in β(0,397824735), ki dajajo zelo majhna odstopanja od prave

vrednosti, dobljene z računanjem kvadratnega korena. Napaka lahko znaša tudi samo 1% (0,1dB).

Kadar tudi množenje s plavajočo vejico ni na razpolago in je časa še manj, uporabimo trik iz celoštevilčnega zapisa: namesto množenja s plavajočo vejico raje množimo s faktorji, ki so dvojiškemu zapisu prijazni, na primer $\alpha = {}^{15}/_{16}$ in $\beta = {}^{15}/_{32}$. S ${}^{15}/_{16}$ množimo tako, da od prvotne vrednosti odštejemo šestnajstino prvotne vrednosti, to pa dobimo tako, da prvotno vrednost pomaknemo za štiri bite v desno. Množenje s plavajočo vejico tako zamenjamo s pomikom v desno in odštevanjem! Podobno množimo s ${}^{15}/_{32}$ tako, da najprej delimo z dva (pomik za eno mesto v desno), nato pa uporabimo prej navedeni postopek. Shema za računanje korena po navedeni metodi je na sliki 15.3. Največja napaka računanja je manjša od 6,25% (0,56dB), povprečna pa znaša 1,79% (0,15dB).



Slika 15.3: Shema za računanje kvadratnega korena za $\alpha = \frac{15}{16}$ in $\beta = \frac{15}{32}$

15.2. Frekvenčna modulacija in demodulacija

Frekvenčno moduliran signal dobimo, če v formuli za harmonski signal spreminjamo frekvenco ω:

$$\cos(\omega t) \rightarrow \cos\big((\omega + x_M)t\big) = \cos(\omega t + x_M t)$$

Frekvenčno modulirani signal lahko generiramo direktno po zgornji formuli, če imamo dovolj hiter računalnik za sprotno izračunavanje funkcije kosinus. V nasprotnem primeru bo morda lažje uporabiti algoritem za oscilator iz poglavja 14.2 ter pri tem sproti z interpolacijo iz v naprej pripravljene tabele določati vrednosti cos φ in sin φ.

15.2.1. Merjenje frekvence signala

Tradicionalno merimo frekvenco signala s štetjem prehodov skozi vrednost nič v časovni enoti. Tako merjenje je ob dovolj veliki časovni enoti v primerjavi s periodo signala lahko dovolj natančno, a je merilni rezultat na voljo samo ob mnogokratnikih časovne enote. Kadar je perioda signala veliko v primeri s časovno enoto raje merimo trajanje ene periode signala in izračunamo frekvenco kot recipročno vrednost

periode. Tudi v tem primeru frekvence signala ne poznamo ves čas, ampak le enkrat v periodi.

15.2.2. Določanje frekvence s pomočjo treh zaporednih izmerkov

Vzemimo, da je signal harmonične oblike in da se njegova frekvenca v času ene periode ne spreminja. Če tak signal vzorčimo z upoštevanjem Nyquistovega kriterija, lahko zapišemo:

$$x_{-1} = A \cdot \sin\big(\omega_{FM}(t - T_v)\big) = A \cdot (\sin \omega_{MF} t \cdot \cos \omega_{FM} T_v - \cos \omega_{MF} t \cdot \sin \omega_{FM} T_v)$$

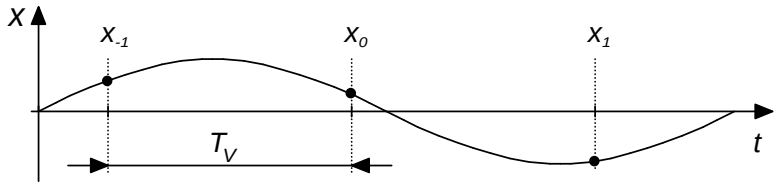
$$x_0 = A \cdot \sin \omega_{FM} t$$

$$x_1 = A \cdot \sin\big(\omega_{FM}(t + T_v)\big) = A \cdot (\sin \omega_{MF} t \cdot \cos \omega_{FM} T_v + \cos \omega_{MF} t \cdot \sin \omega_{FM} T_v)$$

Slika 15.4 kaže položaj vseh treh izmerkov x_{-1} do x_1. Iz zgornjih treh formul izračunamo:

$$\omega_{FM} = \frac{1}{T_v} \ \operatorname{acos} \frac{x_1 + x_{-1}}{2x_0}$$

Formula skriva dve pasti: izmerek x_0 mora biti od nič različen in v eni periodi moramo vzorčiti trikrat, pri tem je dobro, da so vzorci čim bolj razmaknjeni. Obema pastema se ognemo tako, da vzorčimo bistveno pogosteje, kot je to potrebno po Nyquistu in pred računanjem izberemo takšen trojček izmerkov, kjer je obema pogojema zadoščeno. To lahko pomeni, da frekvence ne bomo merili v rednih časovnih intervalih.



Slika 15.4: Trije zaporedni izmerki

15.2.3. Sprotno določanje frekvence

Kadar potrebujemo sprotno informacijo o periodi lahko uporabimo naslednji postopek. Vzorčimo frekvenčno modulirani signal x_{FM}, frekvenca vzorčenja naj bo izbrana ob upoštevanju Nyquistovega kriterija, v nadaljevanju se bo zdelo smiselno vzorčiti s približno štirikratnikom frekvence ω_{FM} opazovanega signala. Zajeti signal popisuje formula:

$$x_{FM}(k) = Ampl \cdot \sin \omega_{FM} k T_v$$

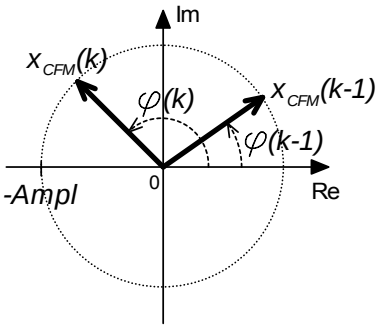
S pomočjo Hilbertovega transforma generirajmo za 90 stopinj premaknjen signal x'_{FM}, tega popisuje formula:

$$x'_{FM}(k) = Ampl \cdot \cos \omega_{FM} k T_v$$

Iz obeh sestavimo kompleksni signal x_{CFM}:

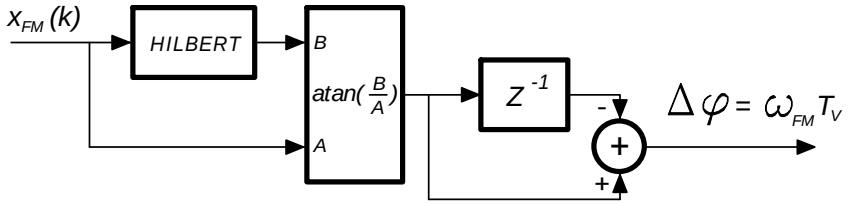
$$x_{C\,FM}(k) = Ampl \cdot e^{i\omega_{FM} k T_v}$$

Kompleksni signal ponazorimo z vektorjem, ki se v kompleksni ravnini ravnini suče v nasprotni smeri urinega kazalca, za vsak naslednji k se kot φ vektorja poveča za $\omega_{FM} T_v$, slika 15.5. Ker je perioda vzorčenja T_v znana, iz spremembe kota izračunamo frekvenco opazovanega signala ω_{FM}.



Slika 15.5: Kompleksni vektor predstavlja merjeni signal

Spremembo kota določimo tako, da sproti za vsak k s pomočjo funkcije atan določimo trenutni kot $\varphi(k)$ kompleksnega vektorja $x_{CFM}(k)$, nato pa od tega odštejemo prejšnji kot $\varphi(k-1)$. Ustrezna shema za določanje trenutne frekvence je na sliki 15.6.



Slika 15.6: Shema za sprotno določanje spremembe kota vektorja v kompleksni ravnini

15.3. Fazna modulacija in demodulacija

15.3.1. Fazna modulacija

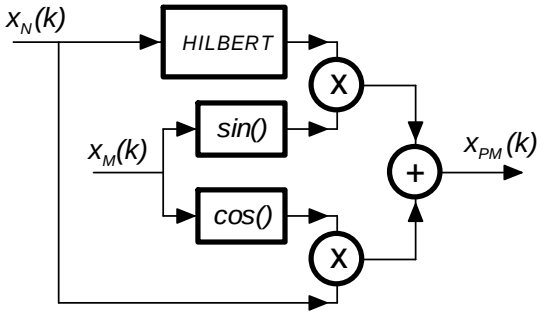
Fazno moduliran signal dobimo, če v formuli za harmonski signal spreminjamo fazo φ v ritmu modulacijskega signala x_M:

$$x_N = \sin(\omega t + \varphi) \quad \rightarrow \quad x_{PM} = \sin(\omega t + x_M) = \sin \omega t \cos x_M + \cos \omega t \sin x_M$$

Kadar potrebujemo fazno moduliran signal, lahko zgornjo formulo implementiramo v računalnik. Vzorčimo (ali generiramo v programu) nosilni signal $\sin \omega t$ in modulacijski signal x_M, potem pa postopamo po shemi 15.7. Nosilnemu signalu s pomočjo Hilbertovega transforma določimo za 90 stopinj zakasnjeno verzijo, izračunamo še vrednosti $\sin x_M$ ter $\cos x_M$. Po množenju in seštevanju dobimo fazno moduliran signal.

Iz zgoraj zapisanega je razbrati, da poljuben signal sestavimo iz uteženih ortogonalnih komponent iste frekvence. Če je faza željenega signala konstantna, lahko uteži izračunamo enkrat v naprej, signala sin ωt in cos ωt pa najlaže dobimo s

pomočjo oscilatorja iz poglavja 14.2. Alternativno lahko komponento cos ωt dobimo tudi s pomočjo Hilbertovega transforma iz sin ωt.



Slika 15.7: Tako generiramo signal x_{PM} s poljubno fazo proti referenčnemu signalu x_N

15.3.2. Sprotno računanje faznega kota – fazna demodulacija

Generiranje fazno moduliranega signala ni naša osnovna naloga. Pri merjenjih fizikalnih veličin je informacija o merjeni veličini pogosto skrita v fazi opazovanega signala. Če znamo pomeriti fazo le tega proti referenčnemu signalu, znamo izluščiti tudi merjeno veličino.

Pri določanju faze potrebujemo referenčni signal x_N, zato je najbolje, da z računalnikom zajemamo tako referenčni kot merjeni signal x_{PM}. Oba signala najprej vodimo na identična pasovno prepustna filtra, tako se znebimo morebitnih motilnih signalov. Za demodulacijo bomo potrebovali še za 90 stopinj zakasnjeni referenčni signal x'_N, ki ga naredimo v računalniku s pomočjo Hilbertovega transforma. Uporabimo nekaj trigonometrije:

$$f = x_{PM} \cdot x_N = \frac{1}{2}(\cos x_M - \cos(2\omega_N t + x_M))$$

$$g = x_{PM} \cdot x'_N = \frac{1}{2}(\sin x_M + \sin(2\omega_N t + x_M))$$

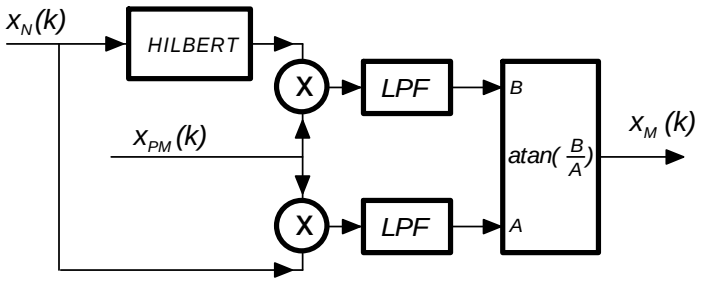
Frekvenco nosilnega signala izberemo tako, da je mnogo večja od frekvence modulacijskega signala. Potem imamo pri zgornjih formulah opravka z dvema komponentama, frekvenca leve komponente je mnogo manjša od frekvence desne komponente. S pomočjo nizko prepustnega filtra izločimo visokofrekvenčni komponenti iz obeh signalov in dobimo:

$$f' = \frac{\cos x_M}{2} \qquad ter \qquad g' = \frac{\sin x_M}{2}$$

Iz tega določimo modulacijski signal oziroma fazni kot med merjenim in referenčnim signalom kot:

$$x_M = \varphi = \operatorname{atan} \frac{g'}{f'}$$

Zaradi kvocienta v argumentu funkcije atan je merilni sistem neobčutljiv na velikosti referenčnega in merjenega signala. Shema za računanje faznega kota med dvema signaloma je na sliki 15.8. Potrebujemo le Hilbertovo transformacijo, dva množilnika, dva nizko prepustna filtra *LPF* in modul, ki računa funkcijo arcus tangens kvocianta argumentov. Fazni kot določamo sproti za vsak izmerek. Ker je frekvenca spreminjanja faznega kota bistveno manjša od frekvence nosilnega signala in s tem tudi od frekvence vzorčenja, lahko povečamo točnost računanja z dodatnim nizko prepustnim filtrom.



Slika 15.8: Shema za določanje faznega kota med dvema signaloma

Pri večini sistemov za zajemanje je treba še upoštevati, da analogno-digitalni pretvornik signala na dveh kanalih zajema izmenoma, zato so izmerki z obeh kanalov časovno premaknjeni za $T_v/_2$.

15.3.3. Računanje faznega kota s križno korelacijo

Kadar ne potrebujemo faznega kota tako pogosto, lahko uporabimo metodo z korelacijo. Pri korelaciji računamo vrednost funkcije, ki je vsota produktov med dvema nizoma, pri tem enega od nizov premikamo:

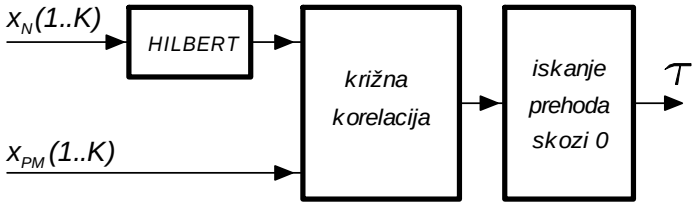
$$Diskretno: \qquad x_{CORR}(\tau) = \frac{1}{K}\sum_{k} x_N(k)x_{PM}(k+\tau)$$

$$Zvezno, periodično: \qquad x_{CORR}(\tau) = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} x_N(t)\cdot x_{PM}(t+\tau)dt$$

Spet zajemamo oba signala x_N in x_{PM}. Ko se nabere primerno število izmerkov, določimo korelacijsko funkcijo med obema nizoma. Ker imata signala isto frekvenco, ima tako frekvenco tudi korelacijska funkcija. Korelacijska funkcija ima največjo vrednost pri tistem τ, kjer sta funkciji poravnani. Poiskati bi torej morali maksimum korelacijske funkcije in od tod odčitati vrednost τ, ki je enaka faznemu kotu med vstopnima nizoma. Žal je iskanje maksimuma pri časovno diskretiziranih funkcijah težka naloga, ki pa se ji lahko izognemo. Uporabimo Hilbertov transform in zakasnimo enega od signalov, na primer x_N, za 90 stopinj; dobimo signal x_n'. Potem je ustrezna

korelacijska funkcija med signaloma x_{PM} in x_n' prav tako premaknjena za 90 stopinj, in iskani fazni kot med obema signaloma se skriva pri prehodu korelacijske funkcije skozi vrednost nič. To pa je mnogo lažje določiti od položaja maksimuma. Potrebno je le poiskati dve sosednji točki na nasprotni straneh aobscisne osi in na primer s pomočjo linearne interpolacije določiti τ, pri katerem ima funkcija $x_{CORR}(\tau)$ vrednost nič. Dodatno poenostavitev predstavlja dejstvo, da v bližini prehoda skozi nič harmonsko funkcijo lahko aproksimiramo s premico, torej je raba linearne interpolacije upravičena.

Na sliki 15.9 je shema za računanje faznega kota s pomočjo korelacijske funkcije.



Slika 15.9: Določanje faznega kota s pomočjo križne korelacije

15.3.4. Računanje faznega kota s prilagajanjem funkcij

Kadar je hitrost spreminjanja faze merjenega signala počasna, lahko uporabimo tudi metodo s prilagajanjem funkcij. Spet vzorčimo referenčni (x_N) in merjeni (x_{PM}) signal, oba imata isto frekvenco. Ko naberemo primerno dolg niz vzorcev (K vzorcev) obeh signalov, se lotimo računanja. To temelji na dejstvu, da lahko sestavimo harmonski signal s poljubnim faznim kotom iz pravilno uteženih harmonskih komponent, ki sta med sabo premaknjeni za 90 stopinj (x_N in x_N', slednjo izračunamo s pomočjo Hilbertove transformacije iz zajetega referenčnega signala).

$$x_{PM} = Ax_N + Bx_N' \qquad \text{pri tem velja:} \quad \sqrt{A^2 + B^2} = 1 \quad \text{in} \quad \varphi = \operatorname{atan} \frac{A}{B}$$

Za zaporedne izmerke lahko sestavimo sistem enačb:

$$\begin{aligned} x_{PM1} &= Ax_{N1} + Bx_{N1}' \\ x_{PM2} &= Ax_{N2} + Bx_{N2}' \\ &\vdots \\ x_{PMK} &= Ax_{NK} + Bx_{NK}' \end{aligned}$$

Ali v matrični obliki:

$$\begin{bmatrix} x_{PM1} \\ x_{PM2} \\ \vdots \\ x_{PMK} \end{bmatrix} = \begin{bmatrix} x_{N1} & x_{N1}' \\ x_{N2} & x_{N2}' \\ & \vdots \\ x_{NK} & x_{NK}' \end{bmatrix} \cdot \begin{bmatrix} A \\ B \end{bmatrix} \qquad \equiv \qquad [X] = [H] \cdot [Y]$$

Rešitev takega sistema lahko bralec poišče drugje, tukaj navajamo le rezultat:

$$[Y] = ([H]^T[H])^{-1} \cdot [X]$$

Iz tega izračunamo še fazni kot kot:

$$\varphi = \text{atan}\,\frac{A}{B}$$

15.3.5. Približek za hitro računanje funkcije atan

Računanje funkcije atan je časovno potratno, kar je še posebej očitno pri šibkejših digitalnih sistemih. Kadar se lahko zadovoljimo s približno izračunanim kotom φ, uporabimo spodnji približek:

$$\varphi = \text{atan}\,\frac{A}{B} \approx \frac{A \cdot B}{B^2 + 0{,}28125 \cdot A^2}$$

Približek je dober v območju:

$$|\varphi| \leq \frac{\pi}{4}$$

Napaka v tem območju je manjša od ±0,26 stopinj. Ker je funkcija atan simetrična, lahko sestavimo naslednjo tabelo za računanje po oktantih:

oktant	formula
prvi in osmi	$\varphi \approx \frac{A \cdot B}{B^2 + 0{,}28125 \cdot A^2}$
drugi in tretji	$\varphi \approx \frac{\pi}{2} - \frac{A \cdot B}{A^2 + 0{,}28125 \cdot B^2}$
četrti in peti	$\varphi \approx \frac{\pi}{2} + \frac{A \cdot B}{B^2 + 0{,}28125 \cdot A^2}$
šesti in sedmi	$\varphi \approx -\frac{\pi}{2} - \frac{A \cdot B}{A^2 + 0{,}28125 \cdot B^2}$

Primerni oktant določimo s pomočjo predznakov vrednosti *A* in *B* ter razmerja A/B.

Kadar računski sistem ne zmore deljenja, lahko uporabimo algoritem CORDIC, njegov opis naj bralec najde v ####.

16. Specializirana sredstva za računanje filtrov

16.1. Digitalni signalni procesor

Digitalni signalni procesor je specializirana verzija procesorja, ki je optimiran za digitalno procesiranje signalov. Optimiramo tako, da pospešimo izvajanje tistih operacij, ki so za digitalno procesiranje podatkov najpogostejše ter poskrbimo za čim hitrejšo dostavo argumentov računanja.

Moderni procesorji v marsičem povzemajo principe, ki so bili razviti za potrebe digitalnega signalnega procesiranja, zato marsikaj od spodaj naštetega najdemo tudi v običajnih procesorjih.

16.1.1. Dolžina registrov

Dolžina registrov je prilagojena preciznosti, s katero naj digitalni signalni procesor računa in širini naslovnega vodila. Pogosto so v teh procesorjih specializirani registri, ki služijo kot kazalci v podatkovni spomin in so sposobni avtomatskega povečevanja brez posega programske opreme.

16.1.2. Množenje in seštevanje

Pri digitalnem filtriranju je osnovna operacija kombinacija množenja in seštevanje, kakršno poznamo iz računanja konvolucije. Zato je izvajanje te operacije (MAC, Multiply - ACumulate) optimirano glede hitrosti izvajanja. Operacija navadno poteka tako, da najprej nastavimo kazalec na polje s podatki in kazalec na polje z utežmi ter označimo register, kamor naj se shranijo akumulirani zmnožki. V naslednjem koraku definiramo še način povečevanja vrednosti kazalcev ter dolžino polj in sprožimo izvajanje operacije MAC, ki se potem samostojno izvede brez dodatnega posredovanja programske opreme digitalnega signalnega procesorja.

16.1.3. Vzporedno izvajanje operacij

16.1.4. Programski in podatkovni spomin

V navadnem procesorju je naslovni prostor navadno en sam in v ta prostor je umeščen podatkovni in programski spomin. Zaradi tega je podatkovno vodilo zelo zasedeno, saj mora za vsako operacijo v procesorju iz programskega spomina najprej

prenesti strojni ukaz in potencialne argumente ukaza, nato pa še podatke iz podatkovnega spomina, na katerih naj se operacija izvede ter na koncu rezultat operacije prenesti nazaj v podatkovni spomin. Tak promet na vodilu resno ovira hitrost izvajanja operacij procesorja, zato je pri digitalnih signalnih procesorjih vodil lahko več.

Ločimo naslovni prostor v naslovni prostor za programski spomin in naslovni prostor za podatkovni spomin. Opravka imamo torej z dvema naslovnima vodiloma za program in ukaze, zato je pričakovati dvakratno hitrost prinašanja ukazov in podatkov iz dveh ločenih spominskih enot. Seveda sta podvojeni tudi podatkovni vodili, eno je namenjeno prinašanju ukazov in njihovih argumentov iz programskega spomina, drugo pa branju in pisanju podatkov v podatkovni spomin.

Nekateri proizvajalci digitalnih signalnih procesorjev gredo še dlje in uporabijo tri ločena vodila, enega za programske ukaze ter dve vodili za argumente, saj na primer v operaciji MAC potrebujemo dva operanda, podatek in utež. Tak je procesor firme $$$Analog Devices…

16.1.5. Vodila

16.2. FPGA (Field Programmable Gate Array)

Pomanjkljivost procesorjev je ravno njihova univerzalnost, zaradi katere je marsikdaj žrtvovana hitrost izvajanja operacij. Vsak ukaz je treba sproti prinašati iz programskega spomina, ga dekodirati in šele zatem izvesti, kar je časovno potratno. Strojna oprema, ki zna brez programa opravljati potrebne operacije, je lahko hitrejša.

V ta namen potrebujemo množico logičnih vezij, povezave med njimi pa prilagodimo potrebnemu zaporedju opravljanja operacij. Do pred kratkim si kaj takega ni bilo mogoče predstavljati, saj so bila logična vezja preveč obsežna za povezovanje. Z uvedbo integriranih vezij FPGA, ki že vsebujejo množico univerzalnih logičnih vezij in jih je mogoče poljubno kombinirati, uporabnik lahko »programira« sestavo in medsebojne povezave v kompleksnem logičnem vezju ter tako dejansko sestavi vezje, ki rešuje eno samo nalogo na optimalen način.

Pri procesorjih smo torej imeli na razpolago univerzalno, že sestavljeno logično vezje, programirali pa smo zaporedje izvajanja operacij v tem univerzalnem vezju. Navodila za izvajanje posamezne operacije je bilo treba sproti prinašati iz programskega spomina.

Pri vezjih FPGA vzpostavljamo povezave med logičnimi elementi, ki opravljajo operacije, potrebne za dosego cilja. Ko so povezave enkrat vzpostavljene, vezje računa ne da bi bilo treba ukaze prinašati, zato je bistveno hitrejše. V posameznem FPGA vezju je lahko do && logičnih vrat (XILINX), povezave med njimi pa opišemo na osebnem računalniku v tekstovni datoteki. Posebni prevajalniki opis prevedejo v tako imenovano JEDEC datoteko, to pa pretočimo v FPGA vezje s pomočjo »programatorja«.

Standard JTAG (»jhsgfjkshgsgvek«) definira potrebne povezave med FPGA (JTAG ukaze ubogajo tudi nekatera druga logična vezja in procesorji, med njimi tudi MSP430) in »programatorjem« ter časovni potek signalov, ki zagotavlja prenos opisa povezav v logično vezje.

Za opisovanje povezav med gradniki v FPGA vezju je na razpolago več standardnih »jezikov«, mi bomo uporabljali VHDL (»«). Opis povezav je strukturiran in sestavljen iz:

A) Opisa signalov, ki jih od zunaj priključimo na vezje FPGA
B) Opisa funkcij gradnikov vezja FPGA
C) Opisa povezav med prej navedenimi gradniki vezja FPGA

Ko je ta datoteka pripravljena, lahko preverimo pravilnost zapisanih funkcij s simulacijo. Pred dejanskim prenosom v vezje FPGA dodamo še opis priključnih nožic na integriranem vezju ter navedemo potrebne napetostne nivoje, vse skupaj še enkrat prevedemo in pretočimo v vezje z »programatorjem« JTAG.

Za zgled je podan opis trivialnega vezja, ki opravlja logično funkcijo:

Pri praktičnem delu pouka bomo nekaj enostavnih sistemov skupaj programirali.

17. Priloge

Priloga 1: Tabela ASCII znakov

Priloga 2: Tabela koeficientov za filter po Čebiševu, nizkoprepustna in visokoprepustna verzija

	0000	0001	0010	0011	0100	0101	0110	0111
0000	NULL CTRL-@	DLE CTRL-@	**SP**	0	@	P	`	P
0001	SOH CTRL-A	DC1 CTRL-@	!	1	A	Q	a	q
0010	STX CTRL-B	DC2 CTRL-@	»	2	B	R	b	r
0011	ETX CTRL-C	DC3 CTRL-@	#	3	C	S	c	s
0100	EOT CTRL-D	DC4 CTRL-@	$	4	D	T	d	t
0101	ENQ CTRL-E	NAK CTRL-@	%	5	E	U	e	u
0110	ACK CTRL-F	SYN CTRL-@	&	6	F	V	f	v
0111	BEL CTRL-G	ETB CTRL-@	'	7	G	W	g	w
1000	BS CTRL-H	CAN CTRL-@	(	8	H	X	h	x
1001	HT CTRL-I	EM CTRL-@	)	9	I	Y	i	y
1010	**LF** CTRL-J	SUB CTRL-@	*	:	J	Z	j	z
1011	VT CTRL-K	ESC CTRL-@	+	;	K	[	k	{
1100	FF CTRL-L	FS CTRL-@	,	<	L	\	l	|
1101	**CR** CTRL-M	GS CTRL-@	-	=	M	]	m	}
1110	SO CTRL-N	RS CTRL-@	.	>	N	^	n	~
1111	SI CTRL-O	US CTRL-@	/	?	O	_	o	**DEL**

*Tabela #: ASCII koda, črka A je na primer predstavljena s kodo 01000001_2 ali 41_{16}. Pri programiranju utegnejo poleg črk in ločil biti pomembne še kode za »LineFeed« **LF** in »CarriageReturn« **CR** ter «Null« in »Space« **SP**.*

Tabela: nizkoprepustni filter, koeficienti po Čebiševu (0.5% valovitosti)				
	2. red	4. red		6. red	
0,01	a0 = 8,663387E-04 a1 = 1,732678E-03 a2 = 8,663387E-04	b1 = 1,919129E+00 b2 = -9,225943E-01	a0 = 4,149425E-07 a1 = 1,659771E-06 a2 = 2,489655E-06 a3 = 1,659770E-06 a4 = 4,149425E-07	Nestabilno! b1 = 3,893453E+00 b2 = -5,688233E+00 b3 = 3,695783E+00 b4 = -9,010106E-01	a0 = 1,391351E-10 a1 = 8,348109E-10 a2 = 2,087027E-09 a3 = 2,782703E a4 = 2,087027E-09 a5 = 8,348109E-10 a6 = 1,391351E-10	Nestabilno! b1 = 5,883343E+00 b2 = -1,442798E+01 b3 = 1,887786E+01 b4 = -1,389914E+01 b5 = 5,459009E+00 b6 = -8,939932E-01
0,025	a0 = 5,112374E-03 a1 = 1,022475E-02 a2 = 5,112374E-03	b1 = 1,797154E+00 b2 = -8,176033E-01	a0 = 1,504626E-05 a1 = 6,018503E-05 a2 = 9,027754E-05 a3 = 6,018503E-05 a4 = 1,504626E-05	b1 = 3,725385E+00 b2 = -5,226004E+00 b3 = 3,270902E+00 b4 = -7,705239E-01	a0 = 3,136210E-08 a1 = 1,881726E-07 a2 = 4,704314E-07 a3 = 6,272419E-07 a4 = 4,704314E-07 a5 = 1,881726E-07 a6 = 3,136210E-08	Nestabilno! b1 = 5,691653E+00 b2 = -1,353172E+01 b3 = 1,719986E+01 b4 = -1,232689E+01 b5 = 4,722721E+00 b6 = -7,556340E-01
0,05	a0 = 1,868823E-02 a1 = 3,737647E-02 a2 = 1,868823E-02	b1 = 1,593937E+00 b2 = -6,686903E-01	a0 = 2,141509E-04 a1 = 8,566037E-04 a2 = 1,284906E-03 a3 = 8,566037E-04 a4 = 2,141509E-04	b1 = 3,425455E+00 b2 = -4,479272E+00 b3 = 2,643718E+00 b4 = -5,933269E-01	a0 = 1,771089E-06 a1 = 1,062654E-05 a2 = 2,656634E-05 a3 = 3,542179E-05 a4 = 2,656634E-05 a5 = 1,062654E-05 a6 = 1,771089E-06	b1 = 5,330512E+00 b2 = -1,196611E+01 b3 = 1,447067E+01 b4 = -9,937710E+00 b5 = 3,673283E+00 b6 = -5,707561E-01
0,075	a0 = 3,869430E-02 a1 = 7,738860E-02 a2 = 3,869430E-02	b1 = 1,392667E+00 b2 = -5,474446E-01	a0 = 9,726342E-04 a1 = 3,890537E-03 a2 = 5,835806E-03 a3 = 3,890537E-03 a4 = 9,726342E-04	b1 = 3,103944E+00 b2 = -3,774453E+00 b3 = 2,111238E+00 b4 = -4,562908E-01	a0 = 1,797538E-05 a1 = 1,078523E-04 a2 = 2,696307E-04 a3 = 3,595076E-04 a4 = 2,696307E-04 a5 = 1,078523E-04 a6 = 1,797538E-05	b1 = 4,921746E+00 b2 = -1,035734E+01 b3 = 1,189764E+01 b4 = -7,854533E+00 b5 = 2,822109E+00 b6 = -4,307710E-01
0,10	a0 = 6,372802E-02 a1 = 1,274560E-01 a2 = 6,372802E-02	b1 = 1,194365E+00 b2 = -4,492774E-01	a0 = 2,780755E-03 a1 = 1,112302E-02 a2 = 1,668453E-02 a3 = 1,112302E-02 a4 = 2,780755E-03	b1 = 2,764031E+00 b2 = -3,122854E+00 b3 = 1,664554E+00 b4 = -3,502232E-01	a0 = 9,086148E-05 a1 = 5,451688E-04 a2 = 1,362922E-03 a3 = 1,817229E-03 a4 = 1,362922E-03 a5 = 5,451688E-04 a6 = 9,086148E-05	b1 = 4,470118E+00 b2 = -8,755594E+00 b3 = 9,543712E+00 b4 = -6,079376E+00 b5 = 2,140062E+00 b6 = -3,247363E-01
0,15	a0 = 1,254285E-01 a1 = 2,508570E-01 a2 = 1,254285E-01	b1 = 8,070778E-01 b2 = -3,087918E-01	a0 = 1,180009E-02 a1 = 4,720034E-02 a2 = 7,080051E-02 a3 = 4,720034E-02 a4 = 1,180009E-02	b1 = 2,039039E+00 b2 = -2,012961E+00 b3 = 9,897915E-01 b4 = -2,046700E-01	a0 = 8,618665E-04 a1 = 5,171199E-03 a2 = 1,292800E-02 a3 = 1,723733E-02 a4 = 1,292800E-02 a5 = 5,171199E-03 a6 = 8,618665E-04	b1 = 3,455239E+00 b2 = -5,754735E+00 b3 = 5,645387E+00 b4 = -3,394902E+00 b5 = 1,177469E+00 b6 = -1,836195E-01
0,20	a0 = 1,997396E-01 a1 = 3,994792E-01 a2 = 1,997396E-01	b1 = 4,291048E-01 b2 = -2,280633E-01	a0 = 3,224554E-02 a1 = 1,289821E-01 a2 = 1,934732E-01 a3 = 1,289821E-01 a4 = 3,224554E-02	b1 = 1,265912E+00 b2 = -1,203878E+00 b3 = 5,405908E-01 b4 = -1,185538E-01	a0 = 4,187408E-03 a1 = 2,512445E-02 a2 = 6,281112E-02 a3 = 8,374816E-02 a4 = 6,281112E-02 a5 = 2,512445E-02 a6 = 4,187408E-03	b1 = 2,315806E+00 b2 = -3,293726E+00 b3 = 2,904826E+00 b4 = -1,694128E+00 b5 = 6,021426E-01 b6 = -1,029147E-01
0,25	a0 = 2,858110E-01 a1 = 5,716221E-01 a2 = 2,858110E-01	b1 = 5,423258E-02 b2 = -1,974768E-01	a0 = 7,015301E-02 a1 = 2,806120E-01 a2 = 4,209180E-01 a3 = 2,806120E-01 a4 = 7,015301E-02	b1 = 4,541481E-01 b2 = -7,417536E-01 b3 = 2,361222E-01 b4 = -7,096476E-02	a0 = 1,434449E-02 a1 = 8,606697E-02 a2 = 2,151674E-01 a3 = 2,868899E-01 a4 = 2,151674E-01 a5 = 8,606697E-02 a6 = 1,434449E-02	b1 = 1,076052E+00 b2 = -1,662847E+00 b3 = 1,191063E+00 b4 = -7,403087E-01 b5 = 2,752158E-01 b6 = -5,722251E-02
0,30	a0 = 3,849163E-01 a1 = 7,698326E-01 a2 = 3,849163E-01	b1 = -3,249116E-01 b2 = -2,147536E-01	a0 = 1,335566E-01 a1 = 5,342263E-01 a2 = 8,013394E-01 a3 = 5,342263E-01 a4 = 1,335566E-01	b1 = -3,904486E-01 b2 = -6,784138E-01 b3 = -1,412021E-02 b4 = -5,392238E-02	a0 = 3,997487E-02 a1 = 2,398492E-01 a2 = 5,996231E-01 a3 = 7,994975E-01 a4 = 5,996231E-01 a5 = 2,398492E-01 a6 = 3,997487E-02	b1 = -2,441152E-01 b2 = -1,130306E+00 b3 = 1,063167E-01 b4 = -3,463299E-01 b5 = 8,882992E-02 b6 = -3,278741E-02
0,35	a0 = 5,001024E-01 a1 = 1,000205E+00 a2 = 5,001024E-01	b1 = -7,158993E-01 b2 = -2,845103E-01	a0 = 2,340973E-01 a1 = 9,363892E-01 a2 = 1,404584E+00 a3 = 9,363892E-01 a4 = 2,340973E-01	b1 = -1,263672E+00 b2 = -1,080487E+00 b3 = -3,276296E-01 b4 = -7,376791E-02	a0 = 9,792321E-02 a1 = 5,875393E-01 a2 = 1,468848E+00 a3 = 1,958464E+00 a4 = 1,468848E+00 a5 = 5,875393E-01 a6 = 9,792321E-02	b1 = -1,627573E+00 b2 = -1,955020E+00 b3 = -1,075051E+00 b4 = -5,106501E-01 b5 = -7,239843E-02 b6 = -2,639193E-02
0,40	a0 = 6,362308E-01 a1 = 1,272462E+00 a2 = 6,362308E-01	b1 = -1,125379E+00 b2 = -4,195441E-01	a0 = 3,896966E-01 a1 = 1,558787E+00 a2 = 2,338180E+00 a3 = 1,558787E+00 a4 = 3,896966E-01	b1 = -2,161179E+00 b2 = -2,033992E+00 b3 = -8,789098E-01 b4 = -1,610655E-01	a0 = 2,211834E-01 a1 = 1,327100E+00 a2 = 3,317751E+00 a3 = 4,423668E+00 a4 = 3,317751E+00 a5 = 1,327100E+00 a6 = 2,211834E-01	b1 = -3,058672E+00 b2 = -4,390465E+00 b3 = -3,523254E+00 b4 = -1,684185E+00 b5 = -4,414881E-01 b6 = -5,767513E-02
0,45	a0 = 8,001101E-01 a1 = 1,600220E+00 a2 = 8,001101E-01	b1 = -1,556269E+00 b2 = -6,441713E-01	a0 = 6,291693E-01 a1 = 2,516677E+00 a2 = 3,775016E+00 a3 = 2,516677E+00 a4 = 6,291693E-01	b1 = -3,077062E+00 b2 = -3,641323E+00 b3 = -1,949229E+00 b4 = -3,990945E-01	a0 = 4,760635E-01 a1 = 2,856381E+00 a2 = 7,140952E+00 a3 = 9,521270E+00 a4 = 7,140952E+00 a5 = 2,856381E+00 a6 = 4,760635E-01	b1 = -4,522403E+00 b2 = -8,676844E+00 b3 = -9,007512E+00 b4 = -5,328429E+00 b5 = -1,702543E+00 b6 = -2,303303E-01

Tabela: visokoprepustni filter, koeficienti po Čebiševu (0.5% valovitosti)				
	2. red	4. red		6. red
0,01	a0 = 9,567529E-01 a1 = -1,913506E+00 b1 = 1,911437E+00 a2 = 9,567529E-01 b2 = -9,155749E-01	a0 = 9,121579E-01 Nestabilno! a1 = -3,648632E+00 b1 = 3,815952E+00 a2 = 5,472947E+00 b2 = -5,465026E+00 a3 = -3,648632E+00 b3 = 3,481295E+00 a4 = 9,121579E-01 b4 = -8,322529E-01		a0 = 8,630195E-01 Nestabilno! a1 = -5,178118E+00 b1 = 5,705102E+00 a2 = 1,294529E+01 b2 = -1,356935E+01 a3 = -1,726039E+01 b3 = 1,722231E+01 a4 = 1,294529E+01 b4 = -1,230214E+01 a5 = -5,178118E+00 b5 = 4,689218E+00 a6 = 8,630195E-01 b6 = -7,451429E-01
0,025	a0 = 8,950355E-01 a1 = -1,790071E+00 b1 = 1,777932E+00 a2 = 8,950355E-01 b2 = -8,022106E-01	a0 = 7,941874E-01 a1 = -3,176750E+00 b1 = 3,538919E+00 a2 = 4,765125E+00 b2 = -4,722213E+00 a3 = -3,176750E+00 b3 = 2,814036E+00 a4 = 7,941874E-01 b4 = -6,318300E-01		a0 = 6,912863E-01 Nestabilno! a1 = -4,147718E+00 b1 = 5,261399E+00 a2 = 1,036929E+01 b2 = -1,157800E+01 a3 = -1,382573E+01 b3 = 1,363599E+01 a4 = 1,036929E+01 b4 = -9,063840E+00 a5 = -4,147718E+00 b5 = 3,223738E+00 a6 = 6,912863E-01 b6 = -4,793541E-01
0,05	a0 = 8,001102E-01 a1 = -1,600220E+00 b1 = 1,556269E+00 a2 = 8,001102E-01 b2 = -6,441715E-01	a0 = 6,291694E-01 a1 = -2,516678E+00 b1 = 3,077062E+00 a2 = 3,775016E+00 b2 = -3,641324E+00 a3 = -2,516678E+00 b3 = 1,949230E+00 a4 = 6,291694E-01 b4 = -3,990947E-01		a0 = 4,760636E-01 a1 = -2,856382E+00 b1 = 4,522403E+00 a2 = 7,140954E+00 b2 = -8,676846E+00 a3 = -9,521272E+00 b3 = 9,007515E+00 a4 = 7,140954E+00 b4 = -5,328431E+00 a5 = -2,856382E+00 b5 = 1,702544E+00 a6 = 4,760636E-01 b6 = -2,303304E-01
0,075	a0 = 7,142028E-01 a1 = -1,428406E+00 b1 = 1,338264E+00 a2 = 7,142028E-01 b2 = -5,185469E-01	a0 = 4,965350E-01 a1 = -1,986140E+00 b1 = 2,617304E+00 a2 = 2,979210E+00 b2 = -2,749252E+00 a3 = -1,986140E+00 b3 = 1,325548E+00 a4 = 4,965350E-01 b4 = -2,524546E-01		a0 = 3,259100E-01 a1 = -1,955460E+00 b1 = 3,787397E+00 a2 = 4,888651E+00 b2 = -6,288362E+00 a3 = -6,518201E+00 b3 = 5,747801E+00 a4 = 4,888651E+00 b4 = -3,041570E+00 a5 = -1,955460E+00 b5 = 8,808669E-01 a6 = 3,259100E-01 b6 = -1,122464E-01
0,10	a0 = 6,362307E-01 a1 = -1,272461E+00 b1 = 1,125379E+00 a2 = 6,362307E-01 b2 = -4,195440E-01	a0 = 3,896966E-01 a1 = -1,558786E+00 b1 = 2,161179E+00 a2 = 2,338179E+00 b2 = -2,033991E+00 a3 = -1,558786E+00 b3 = 8,789094E-01 a4 = 3,896966E-01 b4 = -1,610655E-01		a0 = 2,211833E-01 a1 = -1,327100E+00 b1 = 3,058671E+00 a2 = 3,317750E+00 b2 = -4,390464E+00 a3 = -4,423667E+00 b3 = 3,523252E+00 a4 = 3,317750E+00 b4 = -1,684184E+00 a5 = -1,327100E+00 b5 = 4,414878E-01 a6 = 2,211833E-01 b6 = -5,767508E-02
0,15	a0 = 5,001024E-01 a1 = -1,000205E+00 b1 = 7,158993E-01 a2 = 5,001024E-01 b2 = -2,845103E-01	a0 = 2,340973E-01 a1 = -9,363892E-01 b1 = 1,263672E+00 a2 = 1,404584E+00 b2 = -1,080487E+00 a3 = -9,363892E-01 b3 = 3,276296E-01 a4 = 2,340973E-01 b4 = -7,376791E-02		a0 = 9,792321E-02 a1 = -5,875393E-01 b1 = 1,627573E+00 a2 = 1,468848E+00 b2 = -1,955020E+00 a3 = -1,958464E+00 b3 = 1,075051E+00 a4 = 1,468848E+00 b4 = -5,106501E-01 a5 = -5,875393E-01 b5 = 7,239843E-02 a6 = 9,792321E-02 b6 = -2,639193E-02
0,20	a0 = 3,849163E-01 a1 = -7,698326E-01 b1 = 3,249116E-01 a2 = 3,849163E-01 b2 = -2,147536E-01	a0 = 1,335566E-01 a1 = -5,342262E-01 b1 = 3,904484E-01 a2 = 8,013393E-01 b2 = -6,784138E-01 a3 = -5,342262E-01 b3 = 1,412016E-02 a4 = 1,335566E-01 b4 = -5,392238E-02		a0 = 3,997486E-02 a1 = -2,398492E-01 b1 = 2,441149E-01 a2 = 5,996230E-01 b2 = -1,130306E+00 a3 = -7,994973E-01 b3 = -1,063169E-01 a4 = 5,996230E-01 b4 = -3,463299E-01 a5 = -2,398492E-01 b5 = -8,882996E-02 a6 = 3,997486E-02 b6 = -3,278741E-02
0,25	a0 = 2,858111E-01 a1 = -5,716222E-01 b1 = -5,423243E-02 a2 = 2,858111E-01 b2 = -1,974768E-01	a0 = 7,015302E-02 a1 = -2,806121E-01 b1 = -4,541478E-01 a2 = 4,209182E-01 b2 = -7,417535E-01 a3 = -2,806121E-01 b3 = -2,361221E-01 a4 = 7,015302E-02 b4 = -7,096475E-02		a0 = 1,434450E-02 a1 = -8,606701E-02 b1 = -1,076051E+00 a2 = 2,151675E-01 b2 = -1,662847E+00 a3 = -2,868900E-01 b3 = -1,191062E+00 a4 = 2,151675E-01 b4 = -7,403085E-01 a5 = -8,606701E-02 b5 = -2,752156E-01 a6 = 1,434450E-02 b6 = -5,722250E-02
0,30	a0 = 1,997396E-01 a1 = -3,994792E-01 b1 = -4,291049E-01 a2 = 1,997396E-01 b2 = -2,280633E-01	a0 = 3,224553E-02 a1 = -1,289821E-01 b1 = -1,265912E+00 a2 = 1,934732E-01 b2 = -1,203878E+00 a3 = -1,289821E-01 b3 = -5,405908E-01 a4 = 3,224553E-02 b4 = -1,185538E-01		a0 = 4,187407E-03 a1 = -2,512444E-02 b1 = -2,315806E+00 a2 = 6,281111E-02 b2 = -3,293726E+00 a3 = -8,374815E-02 b3 = -2,904827E+00 a4 = 6,281111E-02 b4 = -1,694129E+00 a5 = -2,512444E-02 b5 = -6,021426E-01 a6 = 4,187407E-03 b6 = -1,029147E-01
0,35	a0 = 1,254285E-01 a1 = -2,508570E-01 b1 = -8,070777E-01 a2 = 1,254285E-01 b2 = -3,087918E-01	a0 = 1,180009E-02 a1 = -4,720035E-02 b1 = -2,039039E+00 a2 = 7,080051E-02 b2 = -2,012961E+00 a3 = -4,720035E-02 b3 = -9,897915E-01 a4 = 1,180009E-02 b4 = -2,046700E-01		a0 = 8,618665E-04 a1 = -5,171200E-03 b1 = -3,455239E+00 a2 = 1,292800E-02 b2 = -5,754734E+00 a3 = -1,723733E-02 b3 = -5,645387E+00 a4 = 1,292800E-02 b4 = -3,394902E+00 a5 = -5,171200E-03 b5 = -1,177469E+00 a6 = 8,618665E-04 b6 = -1,836195E-01
0,40	a0 = 6,372801E-02 a1 = -1,274560E-01 b1 = -1,194365E+00 a2 = 6,372801E-02 b2 = -4,492774E-01	a0 = 2,780754E-03 a1 = -1,112302E-02 b1 = -2,764031E+00 a2 = 1,668453E-02 b2 = -3,122854E+00 a3 = -1,112302E-02 b3 = -1,664554E+00 a4 = 2,780754E-03 b4 = -3,502233E-01		a0 = 9,086141E-05 a1 = -5,451685E-04 b1 = -4,470118E+00 a2 = 1,362921E-03 b2 = -8,755595E+00 a3 = -1,817228E-03 b3 = -9,543712E+00 a4 = 1,362921E-03 b4 = -6,079377E+00 a5 = -5,451685E-04 b5 = -2,140062E+00 a6 = 9,086141E-05 b6 = -3,247363E-01
0,45	a0 = 1,868823E-02 a1 = -3,737647E-02 b1 = -1,593937E+00 a2 = 1,868823E-02 b2 = -6,686903E-01	a0 = 2,141509E-04 a1 = -8,566037E-04 b1 = -3,425455E+00 a2 = 1,284906E-03 b2 = -4,479272E+00 a3 = -8,566037E-04 b3 = -2,643718E+00 a4 = 2,141509E-04 b4 = -5,933269E-01		a0 = 1,771089E-06 a1 = -1,062654E-05 b1 = -5,330512E+00 a2 = 2,656634E-05 b2 = -1,196611E+01 a3 = -3,542179E-05 b3 = -1,447067E+01 a4 = 2,656634E-05 b4 = -9,937710E+00 a5 = -1,062654E-05 b5 = -3,673283E+00 a6 = 1,771089E-06 b6 = -5,707561E-01