10. PRIMER UPORABE MIKROKRMILNIKA

Velika večina mikroprocesorjev, ki so predvideni za manj in srednje zahtevne aplikacije, uporablja le celoštevilski (`integer` v C jeziku) format (npr. 8-bitni Motorola MC68HC11, 16-bitni DSP Texas Instruments 320F240 ali 32-bitni MC68332). Mikroprocesorje z vgrajenim ali dodatnim matematičnim koprocesorjem, ki omogoča bistveno hitrejše računanje s števili zapisanimi v formatu s plavajočo vejico, najdemo šele v višjem cenovnem razredu za zahtevnejše aplikacije, čeprav jim tudi tu konkurirajo sodobni mikroprocesorji brez koprocesorja.

Ena od prednosti procesorjev z matematičnim koprocesorjem je enostavnejša manipulacija s spremenljivkami v programu. Aritmetika s plavajočo vejico omogoča neposredno računanje z realnimi števili, ki lahko ustrezajo vrednostim fizikalnih veličin. Npr. toku 5,87 A v C jeziku lahko priredimo `float` ali `double` spremenljivko **tok** = 5,87. Množenje te spremenljivke s `float` spremenljivko **upornost** = 2,3 (Ω) bo dalo rezultat 13,501, ki ustreza napetosti (npr. `float` spremenljivka **napetost**). Pri tem ne smemo pozabiti, da imajo vhodne (A/D pretvorniki) in izhodne (D/A pretvorniki) enote računalnika na digitalni strani le celoštevilske vrednosti.

Celoštevilska aritmetika izračuna iz predhodnega odstavka ne omogoča na tako preprost način. Eden načinov rešitve tega problema bi lahko bila prilagoditev celoštevilskih spremenljivk vrednostim fizikalnih veličin. Npr. toku 5,87 A priredimo (`short int`, ali `long int`) spremenljivko **tok** = 5870, upornosti pa dodelimo vrednost 2300, torej fizikalne vrednosti množimo s faktorjem 1000 oz. zaokrožimo na tretjo decimalko. Seveda bo treba rezultat izračuna (13501000) deliti z 10^6, da dobimo ustrezno prilagojeno vrednost. Poleg dejstva, da uporabljajo računalniki binarno logiko (torej zahteva deljenje z decimalno vrednostjo 10^6 dodatne ukaze), je očitno tudi, da na ta način ne moremo popolnoma izkoristiti ločljivosti mikroprocesorja (npr. 2^{16} ali 2^{32}). Enako ostaja vprašljivo, kako bi zapisali velika števila, ki presegajo maksimalno vrednost v registru (npr. 1234,939 oz. 1234939 v 16-bitnem registru). Zaradi tega se pri podobnih operacijah raje poslužujemo normiranja.

10.1 Normiranje fizikalnih veličin

V tem poglavju bomo opisali enega izmed načinov normiranja veličin ter ustrezne matematične operacije. V tem in nadaljnjih primerih bomo predpostavili, da je dolžina osnovne besede 16 bitov. To je ločljivost, ki zadošča večini srednje zahtevnih in nekaterim zahtevnejšim nalogam.

Z normiranjem maksimalno izkoristimo številski okvir, ki ga ponuja neki CPU. Običajno moramo pri tem upoštevati logiko dvojiškega komplementa, saj imamo opravka s pozitivnimi in negativnimi vrednostmi spremenljivk[1]. Maksimalni pozitivni vrednosti pri 16-bitni CPU bo

[1] Izjema so nekatere veličine, ki jim lahko dodelimo le pozitivne vrednosti. Npr. kot rotorja motorja lahko vedno zapišemo s pozitivnimi vrednostmi od 0 do 2π, kar bi ustrezalo številskem področju od 0000_{HEX} do $ffff_{HEX}$ (v 16-bitnem sistemu). Negativne kote lahko vedno ponazorimo s pozitivnimi.

potemtakem lahko ustrezala vrednost $7fff_{HEX}$ (2^{15}-1 = 32767_{DEC}), najmanjša negativna vrednost pa je 8000_{HEX} (-2^{15} = -32768_{DEC})[2].

Normiranje je postopek, v katerem neki vrednosti fizikalne veličine (npr. toku, napetosti, hitrosti) dodelimo referenčno številčno vrednost, s katero bo operiral procesor. Izbira norme je odvisna od veliko faktorjev: možnih minimalnih in maksimalnih vrednosti fizikalne veličine, ločljivosti, koherentnosti z normami ostalih veličin itd. Ponavadi pri normiranju izberemo nazivno (npr. nazivni tok motorja) ali maksimalno možno vrednost (npr. kratkostični tok v motorju) fizikalne veličine. Katera heksadecimalna vrednost bo ustrezala normi v mikroprocesorju, je odvisno tudi od konkretnega problema, ki ga rešujemo. Seveda želimo številski obseg procesorja popolnoma izkoristiti, zato načeloma priredimo normi čim višjo vrednost. Na ta način bomo lahko računali tudi z vrednostmi, ki so bistveno manjše od normirane.

Norma običajno ustreza nazivni vrednosti, saj so tudi podatki o fizikalnih veličinah reguliranca (npr. motorja) podani za nazivno obratovanje (glej tudi zgled v pogl. 10.3.4). Pri tem pa je treba zelo paziti, da pri vrednostih fizikalnih veličin, ki so večje od norme, ne pride do presežka (overflow). Temu se lahko izognemo že pri izbiri norme, ki ustreza maksimalni vrednosti, to pa lahko pripelje do drugih nezaželenih posledic (npr. maksimalni vrednosti oz. normi toka ne ustreza nujno maksimalna vrednost ali norma napetosti).

Pri normiranju na nazivno vrednost ustreza normi heksadecimalni ekvivalent, ki je manjši od maksimalne možne vrednosti (od -32768 do +32767). Na ta način se tudi pri nadnazivnih vrednostih fizikalne veličine izognemo presežku. Zgled v pogl. 10.3 pojasnjuje praktične probleme pri normiranju.

Glede na to, da je človeku bližje računanje z realnimi decimalnimi števili, običajno med preračunavanjem iz fizikalne vrednosti v heksadecimalno vpeljemo še pomožni zapis, kjer vrednost “-1” ustreza negativni vrednosti norme, “+1” pa pozitivni. Polovični vrednosti norme ustreza “+0,5” itd. (takšen prikaz bomo zaradi ločevanja od navadnega zapisa označevali z narekovaji).

Primer:

Nazivna vrednost toka, ki teče v motor je 33 A, ki naj mu ustreza heksadecimalni ekvivalent 4000_{HEX}. Tok 3 A je “0,0909” norme, torej:

33 A $\rightarrow$ “+1” $\rightarrow$ 16384_{DEC} $\rightarrow$ 4000_{HEX},

Šestnajstiško vrednost, ki ustreza toku 3 A, pa dobimo takole:

3 A $\rightarrow$ “+0,0909” $\rightarrow$ $0{,}0909 \cdot 16384_{DEC} \approx 1489_{DEC}$ $\rightarrow$ $5d1_{HEX}$.

Na ta način se izognemo neposrednemu računanju s šestnajstiškimi števili; pretvorbo v ta format opravimo šele iz decimalnega zapisa.

[2] Pri 8-bitnih CPU-jih bi se vrednost gibala med 80_{HEX} in $7f_{HEX}$, pri 32-bitnih pa med 80000000_{HEX} in $7fffffff_{HEX}$.

S pravilno izbiro digitalnih ekvivalentov norm vseh nastopajočih veličin lahko popolnoma izkoristimo številski obseg procesorja. Dodatna prednost se kaže tudi v koherentnosti z veličinami, ki jih nismo neposredno normirali.

Primer:

Kot bomo videli v praktičnem zgledu, normiramo pri motorju tri osnovne veličine (napetost, tok in hitrost rotorja) na nazivne vrednosti, ki so podane na napisni tablici ali smo jih izmerili. S tem so enoumno podane tudi norme parametrov (npr. upornosti, induktivnosti, časovne konstante). Hkrati bodo tudi rezultati enačb normirani. Npr. fluks motorja izračunamo iz enačb, ki vsebujejo prej navedene vrednosti. Če je rezultat tega izračuna "0,7" pomeni, da je trenutno motor magneten s 70 % nazivnega fluksa.

10.1.1 Matematične operacije med normiranimi veličinami

10.1.1.1 Seštevanje in odštevanje

Pri izvajanju operacij med normiranimi celoštevilskimi veličinami v mikroprocesorskem sistemu moramo biti zelo previdni. Enostavne matematične operacije, kot so npr. seštevanje in odštevanje ter logične povezave običajno potekajo brez težav. Edina resnejša nevarnost leži v možnosti preseganja področja osnovne besede.

Primer:

Zaradi enostavnosti predpostavimo, da so veličine normirane na maksimalno možno vrednost, torej "+1" ustreza $+32767_{DEC}$ ($7fff_{HEX}$), "-1" pa -32768_{DEC} (8000_{HEX}). Rezultat seštevanja pozitivnih vrednosti 5555_{HEX} ("0,667" = 21845_{DEC}) in 3000_{HEX} ("0,375" = 12288_{DEC}), npr. z ukazom `ADD`, bo dalo 8555_{HEX} (kar je v 16-bitnem računalniku negativna vrednost: -31403_{DEC}), namesto $+34133_{DEC}$ ("1,045")[3]. Dobljena vrednost je logična, saj bi pričakovani rezultat presegal maksimalno dovoljeno vrednost. Računalnik se bo odzval s setiranjem "overflow" bita (OV ali V) v statusnem registru. Previdnost je potrebna tudi zaradi tega, ker je potek izračuna binarnih zapisov povsem regularen in jo bo načeloma računalnik izvedel brez zapletov (razen postavljenega bita V).

10.1.1.2 Množenje

Pri operacijah množenja in deljenja je potrebna dodatna pozornost. Množenje dveh števil, ki sta manjši ali enaki normi (≤ "1"), ima vedno kot rezultat število, ki je manjše ali enako vrednosti obeh vhodnih števil.

Npr. rezultat množenja normiranih števil iz prejšnjega primera bo:

"0,667" ·"0,375" = "0,25". (10. 1)

[3] Majhna razlika med seštevkom normiranih vrednosti in normiranim rezultatom je posledica zaokroževanja.

Poglejmo, ali sledi ta rezultat tudi kot posledica ukaza za 16-bitno množenje v 32-bitnem mikroprocesorju MC68332, npr. z ukazom `MULS.W D0,D2`. Operanda (spodnjih 16 bitov 32-bitnega registra) se nahajata v D0 in D2, rezultat (32 - bitni) pa v registru D2. Pri D0 = 5555_{HEX} in D2 = 3000_{HEX} bo rezultat D2 = $ffff000_{HEX}$, torej veliko večji od maksimalne vrednosti, ki jo dolžina besed operanda dovoljuje. Kako ta rezultat povezati s pričakovanim rezultatom v (10. 1)?

Množenje normiranih števil lahko zapišemo na naslednji način:

$(\text{“0,667”} \cdot 2^{15}) \cdot (\text{“0,375”} \cdot 2^{15}) = \text{“0,25”} \cdot 2^{30}$.

Faktor 2^{15} (in ne 2^{16}, kot bi mogoče pričakovali) je namreč posledica dejstva, da smo z upoštevanjem dvojiškega komplementa (“-1” do “+1”) celotno področje 2^{16} razdelili na dve polovici. Na ta način smo MSB uporabili za predznak, ki ne prispeva k ločljivosti. Če želimo, da bo rezultat pravilen, moramo eksponent postaviti na enako vrednost množenca in množitelja. V praksi to pomeni aritmetični pomik (angl. shift) rezultata za 15 bitov v desno (“odrežemo” spodnjih 15 bitov). V našem primeru se vsebina D2 spremeni v $1fff_{HEX}$. Po drugi strani velja iz (10. 1):

$\text{“0,25”} \cdot 2^{15} = 8192_{DEC} = 2000_{HEX}$.

Kot vidimo, sta rezultata praktično enaka. Do možne minimalne razlike pride iz dveh razlogov:

1. Normiranje oz. pretvarjanje iz realnih v cela števila nujno prinese nek pogrešek.
2. Pri normiranju (od “-1” do “+1”) smo predpostavili, da sta negativno in pozitivno območje simetrični. V resnici pa je maksimalna negativna celoštevilska vrednost vedno za 1 večja od maksimalne pozitivne zaradi prištevanja vrednosti “0” k pozitivnemu področju (glej pogl. 1.2).

Zgoraj opisana naloga je lažja, ko imamo opravka z množenjem dveh nepredznačenih (angl. unsigned) števil, ki se gibljeta v področju [“0”,“1”]$\cdot 2^{16}$. Takrat moramo iz rezultata [“0”,“1”]$\cdot 2^{32}$ izločiti le spodnjih 16 bitov. Pri 16-bitnih CPU-jih je to še lažje doseči. Tam je produkt registrov sestavljen iz dveh zaporednih registrov. Rezultat bo avtomatično v tistem registru, ki vsebuje besedo zmnožka z višjo težo.

10.1.1.2.1 Množenje z normiranimi vrednostmi, ki so večje od 1

Pri računanju z normiranimi števili pride včasih do potrebe po množenju s faktorjem, ki je večji od “1”, npr.

“0,12” · “-3,65”.

V primeru izbire norme, ki ustreza maksimalni vrednosti registra, zadnjega podatka ne moremo zapisati v enem samem registru, saj presega njegovo maksimalno območje. Zato je množenju treba pristopiti nekoliko drugače.

Zaradi binarne narave organizacije registrov je takšne podatke smiselno prikazati v obliki, ki je produkt normiranega števila od "-1" do "+1" (mantise) in najmanjšega mnogokratnika števila 2 (eksponenta), ki še dovoljuje zapis mantise v normirani obliki[4], npr.

"-3,65" = "-0,9125" · 2^2.

Na ta način način smo zahtevano operacijo prevedli v obliko:

"0,12" · "-0,9125" · 2^2.

Takšno množenje je sestavljeno iz dveh korakov:

1. Standardno množenje dveh normiranih števil.
2. Dodatno množenje z mnogokratnikom binarne baze. To najlažje naredimo z že znano operacijo pomika (shift) rezultata prvega koraka v levo (v tem primeru za dve poziciji). Ta pomik lahko odštejemo od pomika, ki ga zahteva poravnava pri množenju normiranih števil.

Glede na to, da gre pri tem za množenje s vrednostjo, ki je večja od "1", moramo pri 2. koraku s posebnim algoritmom oz. ukazom preveriti, če je prišlo do presežka.

10.1.1.3 Deljenje

Pri deljenju je potrebno upoštevati še nekatera dodatna pravila in nasvete.

V operacijah deljenja v zbirniku je format deljenca povečinoma večji od formata delitelja in rezultata. Npr. v 16-bitnih procesorjih je deljenec zapisan z 32 biti, deljenec pa s šestnajstimi. Rezultat deljenja je 16-bitno število, prav tako tudi ostanek (glej pogl. 4.2.2). To lahko zapišemo v naslednji obliki: 32/16=16:16. Pri procesorjih, kjer računamo z 32-bitnimi podatki, potemtakem velja: 64/32=32:32.

Očitno je torej pred deljenjem treba deljenec spraviti na višji obseg, ki nam bo omogočil izvesti normiranje. Pri 16-bitnem deljenju transformacijo njegovega formata iz 16 v 32 bitov najlažje dosežemo tako, da mu na levi strani dodamo še eno pomožno besedo (običajno sestavljeno iz ničel).

Primer:

Deljenje "0,15" / "0,45" oz. 1333_{HEX} / 3999_{HEX}

naj bi dalo rezultat "0,333" oz. $2aaa_{HEX}$.

S povečanjem formata deljenca, ki ga zahteva operacija (npr. `DIVS.W d0,d1` v MC68332) bomo izvršili operacijo:

13330000_{HEX} / 3999_{HEX},

[4] Mantisa naj bo čim bližja maksimalni vrednosti, saj le tako ohranjamo največjo možno ločljivost.

rezultata pa bosta:

5555_{HEX} (celoštevilski količnik) in 1333_{HEX} (ostanek).

Količnik ne ustreza pričakovanemu rezultatu, temveč je dvakrat večji zato, ker smo z dodajanjem spodnje besede deljencu izvršili naslednjo operacijo:

$[(\text{“}0{,}15\text{”} \cdot 2^{15}) \cdot 2^{16}] / (\text{“}0{,}45\text{”} \cdot 2^{15}) = \text{“}0{,}333\text{”} \cdot 2^{16} = (\text{“}0{,}333\text{”} \cdot 2) \cdot 2^{15}$,

kar je treba upoštevati pri končnem rezultatu[5].

Najpogostejši napaki, ki se pojavljata pri deljenju, sta presežek in deljenje z ničlo. Do prve pride takrat, ko delimo dejansko večje normirano število z manjšim (ne glede na povečanje formata). Rezultat bo takrat večji od “±1”, kar bo javil bit presežka OV (ali V) v statusnem registru. Če že obstaja potreba po takem deljenju, jo moramo izvesti z upoštevanjem eksponentov (podobno kot pri množenju).

Deljenje z ničlo je veliko nevarnejša napaka, ki ima lahko nepredvidljive posledice, saj je rezultat neskončno število. Procesorji običajno takrat samodejno generirajo ustrezno prekinitev, ki jo programer izkoristi za sprožanje nadaljnjih ukrepov.

Deljenje je običajno časovno zelo dolgotrajna operacija (zlasti pri DSP-jih, kjer je lahko najmanj šestnajstkrat daljša od ostalih), zato se ji izogibamo ali nadomestimo z neko drugo operacijo.

Npr. namesto deljenja spremenljivke *a* s konstanto *konst*:

$$\frac{a}{konst}$$

raje vpeljemo množenje z novo konstanto, ki smo jo že vnaprej izračunali med inicializacijskim postopkom:

$a \cdot konst_1$, kjer je $konst_1 = konst^{-1}$.

Enako se splača pri izrazu

$$\frac{a}{b \cdot c}$$

izračunati produkt imenovalca in šele nato izračunati količnika, namesto dvakratnega deljenja z *b* in nato s *c*.

Pogostokrat imamo opravka s kombiniranimi operacijami množenja in deljenja enako normiranih spremenljivk, npr.

[5] **PAZI!** Povečanje formata z dodajanjem ničel ne prispeva k večji ločljivosti (simbolično): $[(\text{“}0{,}15\text{”} \cdot 2^{15}) \cdot 2^{16}] \neq \text{“}0{,}15\text{”} \cdot 2^{31}$!

$$\frac{a \cdot b}{c}.$$

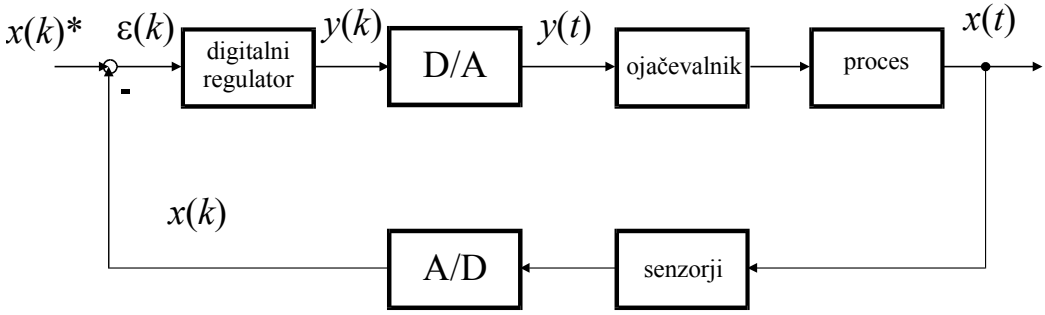
Sedaj že vemo, da imamo v primeru množenja ($a \cdot b$) opravka s 32-bitnim rezultatom. Tega ni potrebno zmanjšati na 16-bitno vrednost, saj bomo ta format uporabili takoj pri deljenju. Na koncu je eventualno potrebna le manjša poravnava v levo ali desno, odvisno od tipa spremenljivk.

10.2 Mikroračunalnik v regulacijskih sistemih

10.2.1 Kratka primerjava analognih in digitalnih regulatorjev

Klasični regulacijski sistemi so vrsto let uporabljali *analogne regulatorje*, ki so sestavljeni iz operacijskih ojačevalnikov, uporov in kondenzatorjev. Z različnimi kombinacijami le-teh je možno zgraditi vse elemente klasične regulacijske tehnike, kot so proporcionalni, diferencialni, in integralni člene ali njihove kombinacije, nizko- in visokopasovni filtri itd. Po definiciji sta frekvenčni pas in ločljivost signalov pri takšnih analognih vezjih visoka. Po drugi strani so glavne pomanjkljivosti analognih rešitev nefleksibilnost, visoka cena elementov, kot so množilniki, omejenost realizacije nekaterih funkcij, nezmožnost shranjevanja in numerične obdelave rezultatov itd.

Glavni sestavni del sodobnih *digitalnih regulatorjev* so mikroprocesorski elementi, kar omogoča veliko fleksibilnost (funkcijo regulatorja takoj spremenimo z drugačnim programom), bistveno manjšo občutljivost na motnje, izvajanje zapletenih algoritmov itd. Slika 10. 1 kaže osnovno blokovno shemo regulacijskega kroga z digitalnim regulatorjem



Slika 10. 1: Regulacijski sistem z digitalnim regulatorjem

Glavna pomanjkljivost digitalnih regulatorjev je v omejeni ločljivosti, kar v določenih primerih pripelje do resnih numeričnih problemov, ter omejen frekvenčni pas. Tabela 10.1 kaže primerjalne karakteristike obeh pristopov [1].

Regulator	Prednosti	Pomanjkljivosti
Analogni	Široko frekvenčno področje Visoka ločljivost Enostavnost načrtovanja	Staranje Temperaturni drift Fiksna (ožičena) struktura Primernost le za manj zahtevne naloge
Digitalni	Programirljive rešitve Neobčutljivost na motnje Aplikacija sodobnih algoritmov Možnost dodatnih nalog	Numerične težave Običajno potreba po uporabi visoko zmogljivih procesorjev Zapletena gradnja

Tabela 10.1: Primerjava analognih in mikroprocesorskih (digitalnih) regulatorjev

10.2.2 Problemi pri izbiri mikroprocesorskih regulatorjev

Večina digitalnih regulatorjev uporablja celoštevilsko aritmetmetiko, kjer je prikaz velikosti signala ali ojačenja možen le s končno ločljivostjo (npr. 8, 16 ali 32 bitov). Za prilagajanje vseh veličin temu področju je potrebno skaliranje oz. normiranje. Efekti končne dolžine besede (angl. finite wordlength) se kažejo v obliki *diskretizacijskega* ali *kvantizacijskega šuma* (angl. quantization noise), ki lahko privedejo celo do nestabilnosti celotnega sistema [1].

Druga posledica končne dolžine besede se kaže v *pogrešku zaokroževanja* (angl. roundoff error). Med zajemanjem in izdajanjem končnih vrednosti je običajno veliko aritmetičnih operacij. Zaradi končne dolžine besede vnese vsaka od njih določen pogrešek, ki se po vsakem ukazu kumulira. Npr. rezultat množenja v 32-bitnem procesorju je običajno shranjen v 64-bitni obliki (pri 16-bitnih pa $16 \cdot 16 = 32$), ker pa ga moramo uporabiti tudi pri nadaljnjih operacijah, je potrebno skrajšanje na osnovni format 32 bitov. To pomeni, da smo izgubili spodnjih 32 bitov. Nadalje, ob povečanju vrednosti registra celoštevilskih procesorjev nad maksimalno vrednost pride do presežka, ko se spremeni predznak. Poseben problem pri normalizaciji je ravno preprečevanje presežka, saj moramo veličine normirati tako, da tudi pri največji vrednosti do tega ne pride. Če pri tem za nazivne vrednosti uporabimo le spodnji del besede (npr. 12 bitov pri formatu f4.12, glej pogl. 10.3.4.2), smo ločljivost še dodatno zmanjšali.

Zakaj se potem pri izbiri digitalnih regulatorjev za aplikacije v močnostni elektroniki ne odločamo za procesorje s plavajočo vejico, kjer ni zgoraj navedenih negativnih učinkov? Osnovni razlog je cena! Namen proizvajalcev je ponuditi procesorje, ki jih bomo lahko uporabili v številnih aplikacijah, kjer lahko cena procesorja odločilno vpliva na ceno končnega izdelka (npr. pralni stroji, klimatske naprave itd.). Multiprocesorski sistemi z vgrajenim matematičnim koprocesorjem so enostavno predragi in je njihova uporaba upravičena le pri reševanju zapletenih problemov.

10.3 Zgled zajemanja veličin, normiranja in realizacije nekaterih funkcij

Kot osnovo za ilustracijo aplikacije mikroprocesorskega sistema bomo vzeli vektorsko regulacijo pogona z asinhronskim motorjem (v nadaljevanju AM). V njej so obrazloženi

posamezni problemi, ki jih srečujemo v večini mikroračunalniških sistemov za regulacijo v realnem času:

1. Najkrajše (električne) časovne konstante konstante sistema so od nekaj ms do nekaj deset ms. Zadosti dobra izbira časa (intervala) vzorčenja je $T_V \leq 1$ ms (glej tudi pogl. 10.3.3). Iz tega sledi zahteva po uporabi hitrega mikroprocesorja, ki bo lahko celotni regulacijski program izvršil znotraj intervala vzorčenja (npr. MC68332 ali DSP320F240). Zaradi konkurenčnosti končnega izdelka si ne smemo privoščiti "predobrega" in s tem tudi dragega mikroprocesorja (cene procesorjev za to aplikacijo so običajno do nekaj deset $).
2. V programu so prisotni različni splošno uporabni algoritmi (npr. nizkopasovni filtri - členi prvega reda, regulatorji, trigonometrične funkcije, v nekaterih primerih pulzno-širinska modulacija napetosti itd.).
3. Mikroračunalniški sistem vsebuje različne vhodno/izhodne enote (A/D in D/A pretvornike, vezja za zajemanje pulzov iz inkrementalnega dajalnika, binarne vhode in izhode).
4. Procesni mikroračunalniški sistem mora imeti možnost komunikacije z nadrejenim računalnikom (običajno osebnim računalnikom prek serijskega protokola EIA-232).

V tem podpoglavju bomo obdelali naslednje teme:

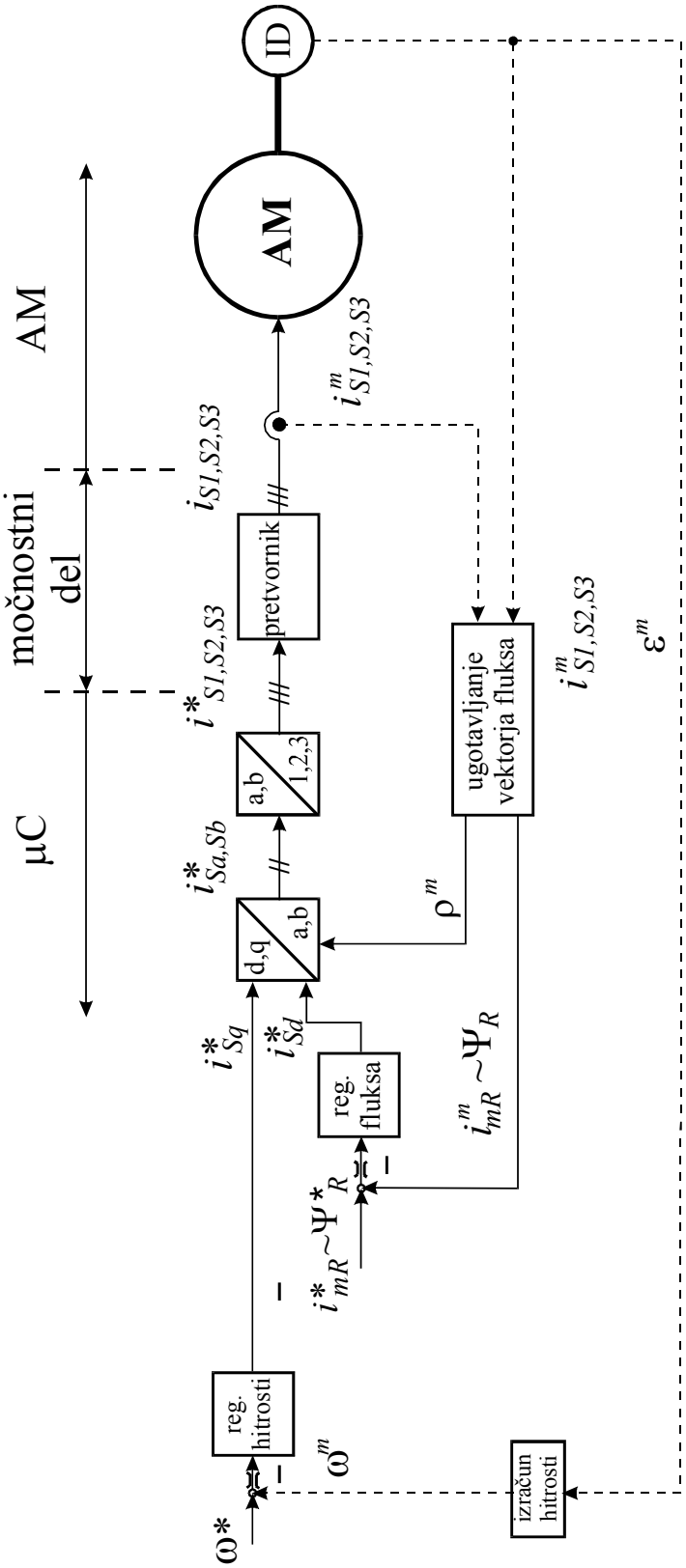
- Opis regulacijske naloge.
- Arhitektura mikroračunalnika.
- Struktura programa.
- Normiranje veličin in karakteristične matematične operacije.
- Realizacija nekaterih splošno uporabnih funkcij.

10.3.1 Opis regulacijske naloge

Blokovno shemo regulacije hitrosti in/ali kota zasuka AM kaže slika 10. 2. Tukaj se ne bomo ukvarjali s podrobnim opisom fizikalnega ozadja celotnega problema, saj je to opisano v literaturi [npr. 2], temveč bomo obravnavali le posamezne splošno uporabne postopke in segmente algoritma. Videli bomo, da gre za dokaj zapleteno regulacijsko nalogo, ki jo mora mikroračunalnik izvajati sproti, v realnem času. Naj omenimo, da je to le eden izmed možnih pristopov k reševanju dane regulacijske naloge.

Pogon je sestavljen iz treh osnovnih delov (glej tudi sliko 10. 5):

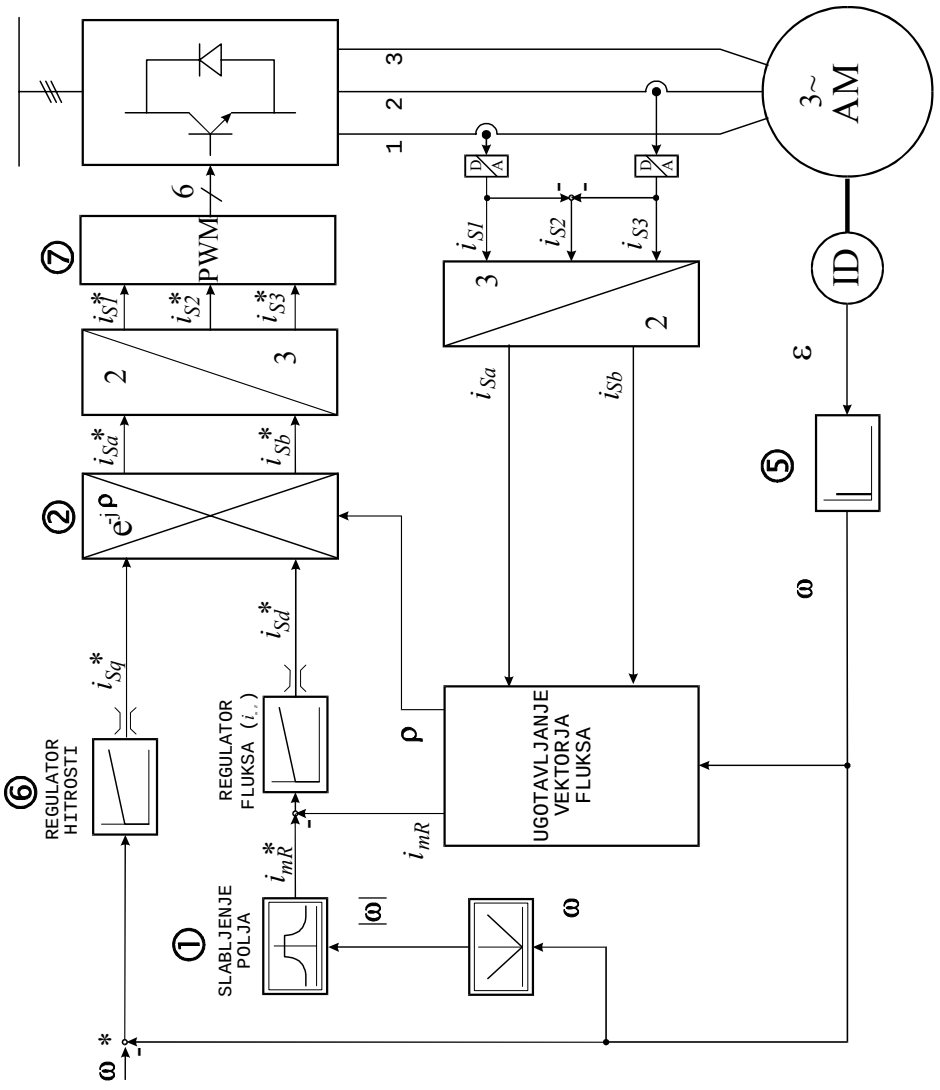
1. Reguliranec je **asinhronski motor**, običajno s kratkostično kletko (indukcijski motor).
2. **Procesni mikroračunalnik** je zadolžen za regulacijo toka ali napetosti, s katero napajamo motor. Pri tem je so potrebne meritve nekaterih veličin iz motorja (npr. napetosti, toka ter kota zasuka rotorja oz. njegovega odvoda - kotne hitrosti).
3. **Močnostni pretvornik** je v bistvu ojačevalnik, ki izhodne signale pretvornika pretvori v dejanske napetosti ali tokove na statorskem navitju motorja. V našem primeru imamo opravka s tokovnim virom, kjer mikroračunalnik narekuje obliko toka. V praksi je pogosto za to zadolžena t.i. pulzna-širinsko modulacija (angl. Pulse Width Modulation - PWM), kjer računalnik generira pulze za proženje tranzitorjev v pretvorniku po določenem algoritmu (pogl. 10.3.5.7).



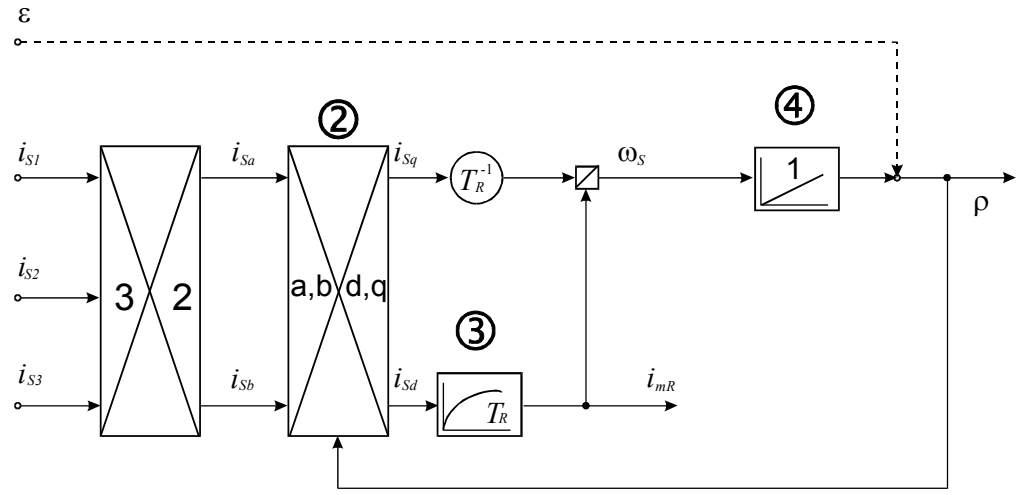
Slika 10. 2: Načelna blokovna shema vektorske regulacije asinhronskega motorja

Kratek opis naloge:

Mikroračunalnik primerja želeno vrednost vrtilne hitrosti (ω*) z dejansko (ω^m). Le-to dobimo z odvajanjem kota rotorja, ki ga merimo s pomočjo inkrementalnega dajalnika (ID - glej tudi pogl. 12.5.2), po času. Razlika hitrosti je vhodna veličina v regulator (običajno PI), izhodna veličina pa tok i_{Sq}*, ki je proporcionalen električnemu momentu. Za regulacijo fluksa motorja je zadolžen drugi regulator (prav tako PI), na katerega pripeljemo razliko med želenim in dejanskim fluksom. Slednjega ne moremo neposredno izmeriti, lahko pa ga izračunamo iz enačb v bloku "ugotavljanje vektorja fluksa". Za to potrebujemo informacijo o dejanskih statorskih tokih in trenutni poziciji rotorja. Izhodno veličino iz regulatorja fluksa i_{Sd}* skupaj z i_{Sq}* pripeljemo na transformacijski blok (množenje in trigonometrične funkcije), katerega rezultat sta *a* in *b* komponenta vektorja statorskega toka v fiksnem koordinatnem sistemu. Po še eni transformaciji (množenje in seštevanje) dobimo tri želene statorske toke i_{S1}*, i_{S2}* in i_{S3}*. To so hkrati vhodne vrednosti v tokovni močnostni pretvornik. Funkcije, ki jih mora izvajati mikroračunalnik, bodo jasnejše, če si ogledamo podrobnejši regulacijski shemi na slikah 10. 3 in 10. 4.



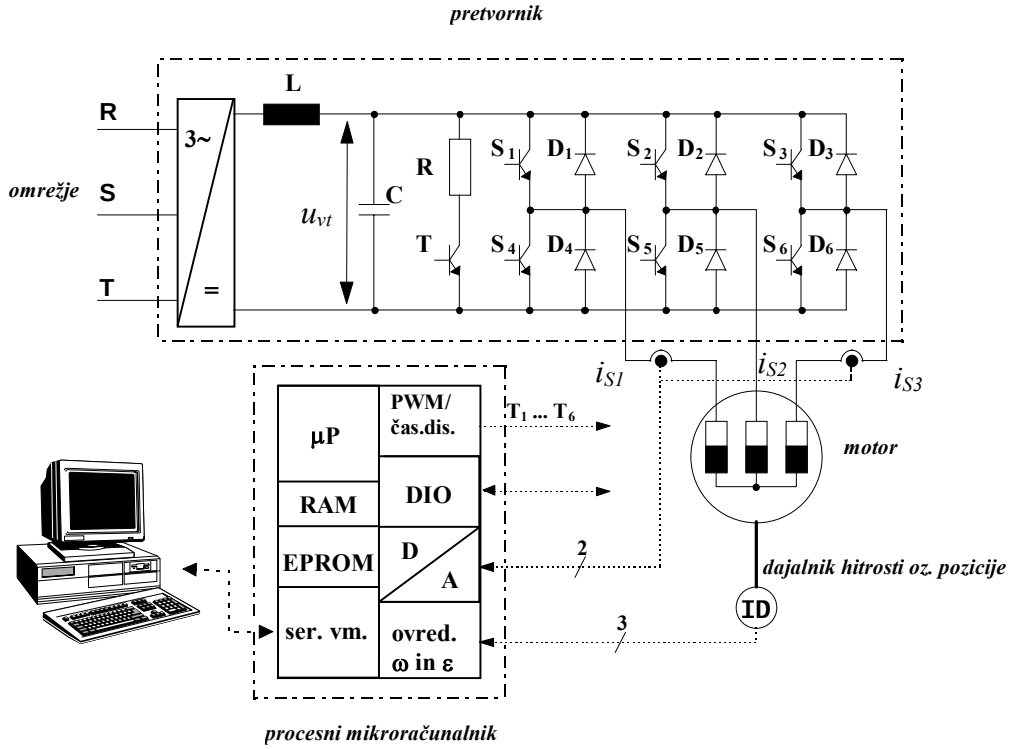
Slika 10. 3: Podrobnejša blokovna shema regulacije AM



Slika 10. 4: Podrobna blokovna shema bloka "ugotavljanje vektorja fluksa" s slike 10. 3 za dvopolni stroj

10.3.2 Arhitektura mikroračunalnika

Slika 10. 5 kaže načelno zgradbo mikroračunalnika za izvajanje regulacije asinhronskega motorja.



Slika 10. 5: Karakteristična konfiguracija modernega pogona z asinhronskim motorjem

Osnovne komponente mikroračunalnika so:

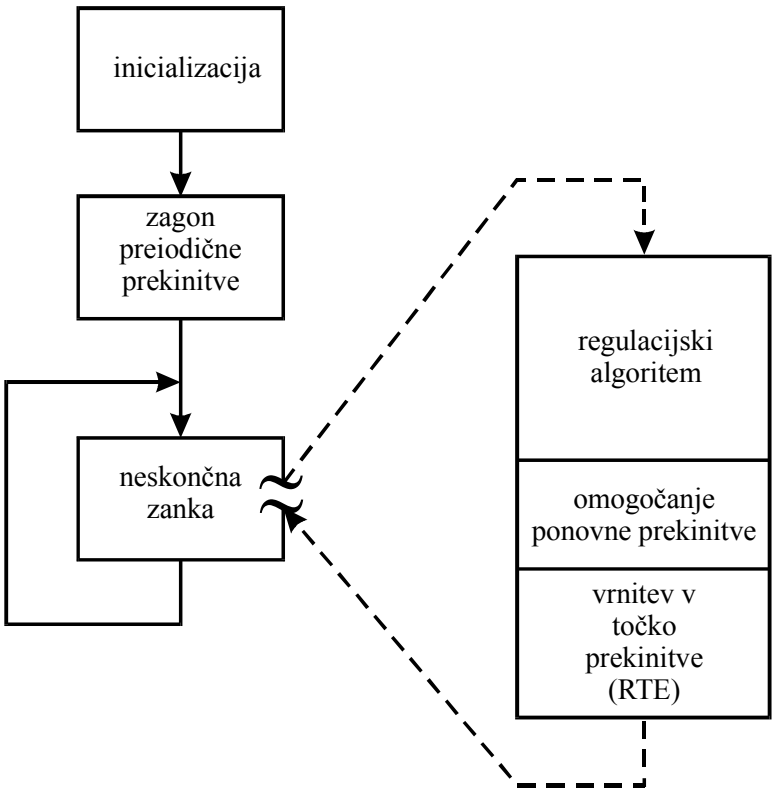
1. **Mikroprocesor (μP)**.
2. **Pomnilnik** je potreben za shranjevanje regulacijskega programa in tabel (običajno bralni pomnilnik, npr. EPROM ali Flash EEPROM) ter spremenljivk (bralno-pisalni pomnilnik - RAM).
3. **Komunikacijska enota** (običajno serijski vmesnik EIA-232) skrbi za povezovanje z *nadrejenim mikroračunalnikom* (npr. osebni računalnik - PC), katerega glavna naloga je določanje načina delovanja procesnega računalnika, parametrov in želenih vrednosti, kakor tudi prikaz tekočih spremenljivk. V primeru kompleksnejših pogonov je naloga PC-ja hierarhično organiziranje in koordiniranje več procesnih računalnikov.
4. **Digitalne vhodno-izhodne enote (DIO)** so namenjene zajemanju (npr. signalov iz inkrementalnega dajalnika, krmilnih pulzov, posebne tipkovnice) ali oddajanju binarnih signalov (npr. krmiljenje prikazovalnika s tekočimi kristali, proženje močnostnih stikalnih elementov).
5. **PWM enota** je nujna za generiranje prožilnih signalov za tranzistorje, če želimo modulirati izhodno napetost želene oblike. Pri starejših izvedbah je za to zadolžen poseben čip, nekateri novejši mikrokontrolerji (npr. MC68332 ali TMS320F240) pa že vsebujejo lastno PWM enoto.
6. **Analogne vhodno-izhodne enote.** Zajemanje analognih procesnih veličin (kot so npr. toki ali signali iz analognih dajalnikov pozicije), morebitno filtriranje in pretvorba v digitalno obliko, ki je primerna za obdelavo v μC, je naloga analogno-digitalnih (A/D) pretvornikov. V ta namen se pri zahtevnejših sistemih ponavadi uporabljajo 10- (ločljivost $2^{10} = 1024$) ali 12-bitni (ločljivost $2^{12} = 4096$) A/D pretvorniki s kratkim časom pretvorbe (nekaj μs). Z ločljivostjo ne gre pretiravati, saj je njena zgornja meja odvisna od točnostnega razreda merilnikov ter velikosti motenj, ki so zelo pogost pojav pri takšnih sistemih. Zlasti pri zajemanju signalov iz merilnika hitrosti lahko uporabimo manj zmogljive 8-bitne pretvornike. Dodaten razlog za pravilno oceno potrebne zahtevnosti pretvornikov je cena, ki z višanjem zmogljivosti strmo narašča. Digitalno-analogni (D/A) pretvorniki imajo dvojno funkcijo:

    1. Pri časovno-diskretnem pretvorniku je njihova naloga pretvorba želenih vrednosti tokov, izračunanih v računalniku, v analogne veličine.
    2. Regulacijski algoritem, ki se izvaja v μC, vsebuje veliko vmesnih spremenljivk, ki jih je zelo težko sproti spremljati (npr. vsakih 500 μs). Njihove trenutne vrednosti najlažje opazujemo tako, da jih preko D/A pretvornika prikažemo na osciloskopu.

10.3.3 Struktura programa

Računalnik rabi nek končni čas za izvajanje regulacijskega programa. Na začetku enega programskega cikla se izmenjajo podatki med periferijo (procesom) in mikroračunalnikom. Zaradi zakonitosti, ki veljajo za diskretne sisteme, je potrebno ta čas (interval ali čas vzorčenja T_V) med dvema zajemanjima vhodnih ali izdajama izhodnih signalov vnaprej fiksno določiti. V praksi ponavadi izberemo čas, ki je vsaj šest do desetkrat krajši od najmanjše časovne konstante sistema[6].

[6] Za razliko od strogih zahtev diskretnih regulacijskih sistemov je pri obdelavi signalov frekvenca vzorčenja vsaj dvakrat večja od najvišje frekvence v sistemu.



Slika 10. 6: Principialna struktura programa

Načeloma je vsak program za regulacijo v realnem času sestavljen iz naslednjih delov (slika 10. 6):

1. **Inicializacija** je del programa, ki se ponavadi izvrši le enkrat. Osnovni namen inicializacije je postavitev začetnih vrednosti spremenljivk ali ugotavljanje začetnega stanja procesa (npr. postopek identifikacije parametrov motorja, ki se izvaja pred zagonom).
2. **Zagon periodične prekinitve**. Od tega trenutka naprej bo tekoči program (glej naslednjo točko) periodično prekinjena s prekinitvenim podprogramom. Perioda prekinitve je enaka času vzorčenja T_V.
3. Po zagonu periodične prekinitve preide izvajanje programa v **neskončno zanko**, ki jo bo ciklično prekinjal prekinitveni podprogram. Znotraj neskončne zanke lahko izvajamo časovno nekritične funkcije, kot je npr. komunikacija z nadrejenim računalnikom oz. uporabnikom, ki lahko procesorju posreduje nove informacije (npr. novo želeno hitrost motorja). Zavedati se moramo, da je programu v zanki na voljo le čas od vrnitve iz prekinitvenega podprograma do naslednje prekinitve, kar pomeni, da ne moremo zagotoviti izvajanja celotnega algoritma v zanki do nastopa naslednje prekinitve. Zato lahko v zanko postavimo le algoritme, ki jih lahko prekinjamo brez posledic.
4. **Glavni regulacijski program** se nahaja v prekinitvenem podprogramu. S tako izbiro delovanja smo omogočili izvajanje začetka regulacijskega programa vsakih T_V. Pri tem pa moramo paziti na to, da s tem nismo zagotovili tudi vedno enakega trajanja samega podprograma; to je odvisno od notranjih pogojnih skokov oz. zank in se lahko med

obratovanjem spreminja[7]. Pomembno je le, da je maksimalni čas trajanja regulacijskega algoritma vedno krajši od periode prekinitve, ker bi v nasprotnem primeru lahko prišlo do prekinjanja samega prekinitvenega podprograma, kar bi imelo nepredvidljive posledice[8].

10.3.4 Normiranje veličin in karakteristične matematične operacije

10.3.4.1 Izračun osnovnih vrednosti

V poglavju 10.1 smo že povedali, da je prvi korak pri normiranju izbira referenčne delovne točke, v kateri so znane vrednosti vseh fizikalnih veličin procesa (npr. nazivno obratovanje ali kratek stik pri motorjih). V našem primeru bomo upoštevali nazivne vrednosti, saj jih je večina dostopna na napisni tablici ali katalogih, ostale pa lahko izračunamo iz obstoječih podatkov. V tabeli (10. 11) so podatki v koherentnih enotah za motor in pretvornik (napajalnik).

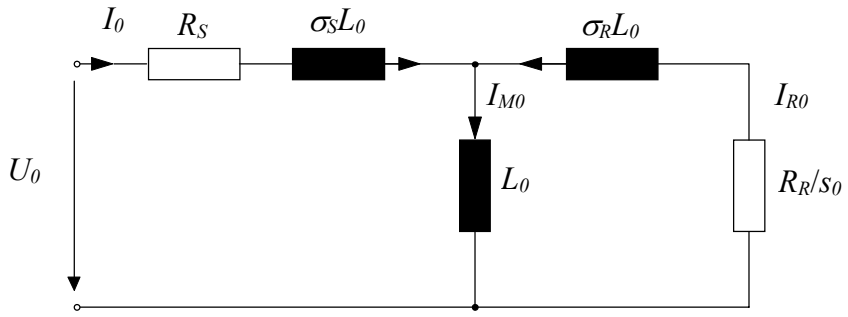
Vezava		zvezda
Nazivna moč	P_0	1,2 kW
Nazivna fazna napetost	U_0	220 V
Nazivna frekvenca (sinhronska vrtilna hitrost)	f_{10} (n_S)	50 Hz (3000 min^{-1})
Nazivna kotna hitrost (nazivna vrtilna hitrost)	ω_0 (n_0)	148,7 s^{-1} (1420 min^{-1})
Nazivni fazni tok	I_0	3,4 A (pri 380 V, 50 Hz)
Število polovih parov	p	2
Faktor moči	cos ϕ	0,86
Statorska upornost	R_S	3,839 Ω
Rotorska upornost	R_R	2,980 Ω
Medsebojna induktivnost	L_0	0,2587 H
Statorska induktivnost	$L_S=L_0(1+\sigma_S)$	0,26627 H
Rotorska induktivnost	$L_R=L_0(1+\sigma_R)$	0,26611 H
Maksimalni tok pretvornika	I_{max}	25 A

Tabela 10.2: Nazivne vrednosti AM

V nazivni delovni točki, v stacionarnem stanju, lahko dogajanja v AM opišemo z *enofazno nadomestno shemo* na sliki 10. 7.

[7] Zaradi tega se načeloma zajemanje vhodnih signalov ter izdajanje izhodnih signalov iz prejšnjega cikla izvaja vedno na začetku prekinitvenega podprograma.

[8] Npr. če je T_V = 1 ms, bo maksimalni čas izvajanja regulacijskega algoritma krajši (npr. 850 μs). Preostalih 150 μs se bo izvajal program v neskončni zanki.



Slika 10. 7: Nadomestna enofazna shema AM v stacionarnem stanju

Najprej bomo normirali osnovne veličine, iz katerih sledijo tudi norme ostalih veličin. V našem primeru bomo izbrali fazno napetost, fazni tok ter kotno frekvenco. Te vrednosti imenujemo *osnovne vrednosti* (angl. base values) [30]. Osnovne vrednosti so enake maksimalni trenutni vrednosti fizikalne veličine v nazivni delovni točki. V tablici so podane efektivne vrednosti toka in napetosti, torej velja:

$I_B = \sqrt{2} I_0 = \sqrt{2} \cdot 3{,}4\ \mathrm{A} = 4{,}8\ \mathrm{A},$
$U_B = \sqrt{2} U_0 = \sqrt{2} \cdot 220\,\mathrm{V} = 311{,}13\ \mathrm{V}$
ter
$\omega_B = 2\pi f_{10} = 2\pi f_{10} = 314{,}159\ \mathrm{rad\ s^{-1}}.$

Z izbiro osnovnih, referenčnih vrednosti zgornjih veličin bomo njihove trenutne vrednosti odslej označevali z relativnimi (angl. pu - per unit: na enoto) vrednostmi, npr. za tok:

$$i_S^{rel} = \frac{i_S}{I_B},$$

kjer je i_S trenutna vrednost statorskega toka. Tako bo relativni tok v trenutku, ko je vrednost sinusnega toka enaka negativni amplitudi nazivnega toka, enak "-1" (glej tudi pogl. 10.1), pri maksimalni možni vrednosti toka, ki jo dovoljuje pretvornik (glej tudi tabelo (10. 11)) pa

$$i_S^{rel} = \frac{i_{\max}}{I_B} = \frac{25\mathrm{A}}{4{,}8\mathrm{A}} = ''5{,}2083''.$$

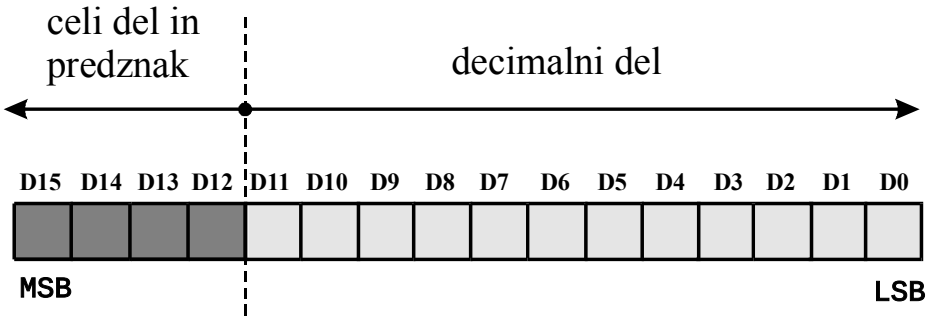
10.3.4.2 Izbira formata zapisa digitalnih vrednosti

Sedaj moramo izbrati še šestnajstiški ekvivalent (*format*) osnovnim normiranim vrednostim, ki smo jih v prejšnjih poglavjih označili z "1". Osnovno besedo razdelimo na, pogojno rečeno, *celi* in *decimalni* del. Pri tem moramo upoštevati ločljivosti osnovne besede (v našem primeru $\pm 2^{15}$) in hkrati paziti, da maksimalna vrednost ne bo presegla številskega področja. V primeru formatiranja toka vidimo, da je maksimalna vrednost 5,2083-krat večja od osnovne.

Za zapis celega števila 5 potrebujemo vsaj tri bite (2^3-1 = 7), glede na to, da je tok lahko pozitiven ali negativen, pa moramo dodati še en bit. Vrednosti "-7" ustreza $8_{HEX} = 1000_{BIN}$, vrednosti "+7" pa $7_{HEX} = 0111_{BIN}$. Za decimalni del ("0" - "1") potemtakem preostaja še 12 bitov, kar pomeni, da je njegova ločljivost

$$\frac{1}{2^{12}} = 0{,}0002414.$$

Takšen format simbolično zapišemo kot f4.12 (slika 10. 8)



Slika 10. 8: Grafična ponazoritev formata f4.12

Pretvorba iz relativne normirane vrednosti v formatu f4.12 poteka torej v dveh delih:

1. Celi del pretvorimo v 4-bitni dvojiški komplement.
2. Decimalni del dobimo tako, da najprej izračunamo desetiški ekvivalent decimalnega dela (x), ki ga nato pretvorimo v šestnajstiško oz. binarno obliko ("x" je decimalni del normirane vrednosti):

$$x = \text{“}x\text{”} \cdot 2^{12}\ (\text{DEC}) = \text{“}x\text{”} \cdot 4096\ (\text{DEC}) \ \rightarrow\ \text{HEX (BIN)}. \qquad (10.\ 2)$$

Oglejmo si nekaj primerov digitalnih ekvivalentov toka:

Dejanska vrednost (A)	Normirana vrednost	Digitalna vrednost (HEX)
-4,8	"-1"	f000
+4,8	"+1"	1000
+25	"5,2083"	5355
+2,4	"0,5"	0800

10.3.4.3 Aritmetične operacije med digitalnimi formatiranimi števili

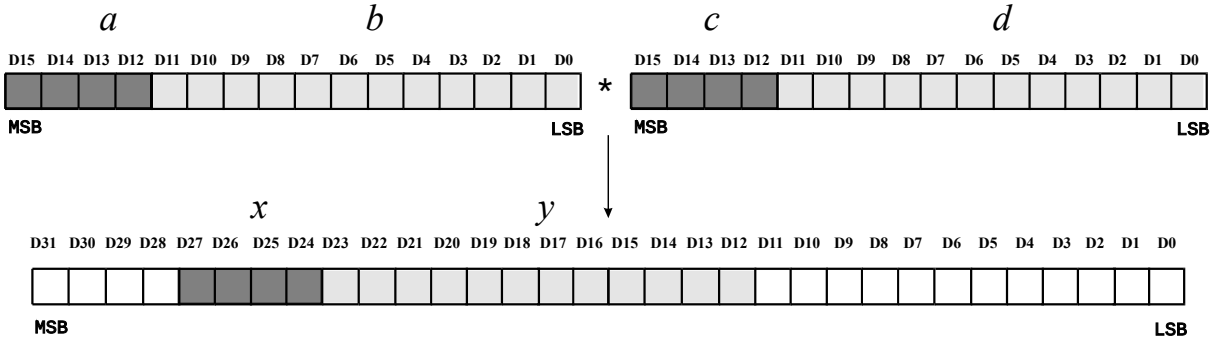
Preverimo, ali logika dvojiškega komplementa še vedno drži: s seštevanjem dveh tokov 25 A in -4,8 A dobimo 20,2 A, seštevanje njunih digitalnih ekvivalentov pa da

5355_{HEX}
$+f000_{HEX}$
$\overline{14355_{HEX}}$

prenos, nepomembno

Če dobljeno število pretvorimo nazaj v relativno vrednost po postopku, ki je inverzen opisanemu v (10. 2), dobimo "4,2083". Decimalni del smo izračunali tako, da smo $355_{HEX} = 853_{DEC}$ delili s 4096_{DEC} ter dobili $0{,}2083_{DEC}$. Dejanska vrednost toka pa je "4,2083"·4,8 A = 20,19984 A, torej izračun drži!

Pri množenju in zlasti pri deljenju je treba biti nekoliko bolj pozoren, saj postopek navadnega množenja, ki smo ga opisali v poglavju 10.1.1.2, ne drži več[9]. Naš cilj je zmnožiti dve števili *a.b* in *c.d* (*a* in *c* sta cela dela, *b* in *d* pa decimalna dela množenca in množitelja) v formatu f4.12 in dobiti rezultat *x.y* v enakem formatu. Slika 10. 9 nazorno kaže format zmnožka.



Slika 10. 9

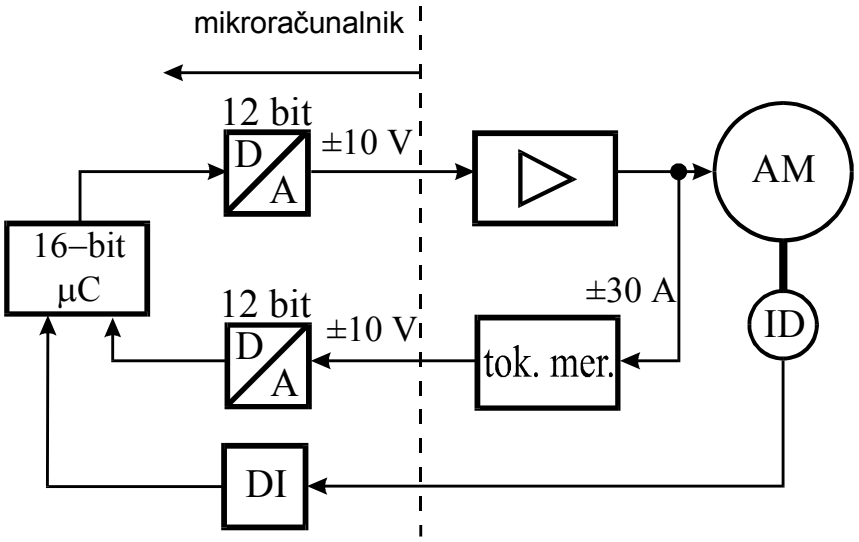
Rezultat v f4.12 bo premaknjen v desno od MSB za 4 bite oz. levo od LSB za 12 bitov. Če hočemo dobiti rezultat v želenem formatu, moramo ustrezno poravnati njegovo vsebino ter upoštevati le zgornjo ali spodnjo besedo tako prilagojenega rezultata. Po premisleku, vidimo, da smo na enak problem naleteli v pogl. 10.1.1.2, kjer smo ob operaciji f1.15·f1.15 dobili rezultat v obliki f2.30. Od tod tudi potreba po premiku za 15 bitov v desno in upoštevanju le LSW.

Naj tukaj opozorimo še na potrebno previdnost pri vseh operacijah med veličinami, ki niso formatirane na enak način. To je sicer dovoljeno, pred vsako operacijo pa moramo skrbno premisliti kako bomo povezali različna formata.

10.3.4.4 Prilagajanje vhodnih in izhodnih vrednosti normam

Nekatere izmed normiranih veličin v mikroprocesorju smo zajeli s pomočjo merilnikov ali pa jih bomo morali posredovati procesu, kjer imamo opravka z dejanskimi fizikalnimi veličinami. Tukaj bomo pokazali le primera zajemanja preko A/D pretvornika (toka) ter diskretnega dajalnika pulzov (kot rotorja), kar kaže slika 10. 10.

[9] Sedaj lahko rečemo, da smo imali tam opravka z množenjem števil v formatu f1.15!



Slika 10. 10: Osnovna aparaturna oprema za regulacijo asinhronskega motorja

10.3.4.4.1 Zajemanje preko A/D in D/A pretvornika

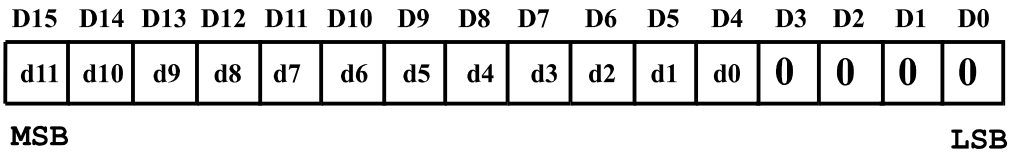
V konkretnem primeru zajemamo tok s tokovnim merilnikom. Vhodnemu območju ±30 A ustreza izhodna napetost ±10 V. To napetostno območje je značilno za A/D pretvornike, zato lahko izhod merilnika priključimo neposredno na pretvornik (ob dodatnih zaščitnih ukrepih, npr. proti prenapetosti). Ločljivost pretvornika naj bo 12-bitna (4096 enot), torej je ločljivost zajemanja napetosti 20 V/4096 = 4,88 mV, ločljivost merjenega toka pa 60 A/4096 = 14,65 mA. Nekatere karakteristične številske vrednosti A/D pretvornika in ustrezne toke kaže tabela 10.3. Pri tem predpostavljamo, da imamo opravka z bipolarnim pretvornikom, torej takim, ki upošteva logiko dvojiškega komplementa (glej tudi pogl. 12.4).

Merjeni tok (A)	Napetost na A/D (V)	HEX 12-bit	HEX 16-bit (leva poravnava)
-30	-10	800	8000
-15	-5	c00	c000
-0.01465	-0.00488	fff	fff0
0	0	000	0000
0.01465	0.00488	001	0010
15	5	3ff	3ff0
30	10	7ff	7ff0

Tabela 10.3: Relacije med merjenimi toki, vhodnimi napetostmi in digitalnimi vrednostmi 12-bitnega pretvornika

Z ozirom na to, da se format pretvornika (12-bitov) ne ujema s formatom mikroračunalnika (16 bitov), je potrebna ustrezna prilagoditev. Če hočemo, da bo logika dvojiškega komplementa veljala pri obeh formatih, moramo bite pretvornika "*poravnati v levo*" (angl. left justify) znotraj mikroračunalniške besede (tabela 10.3). V tem primeru se bosta MSB

obeh formatov ujemala, prazne lokacije pa bodo zapolnjene z ničlami (slika 10. 11). Poravnavo lahko izvršimo softversko (pomik za štiri bite v levo) ali hardversko (z ustrezno povezavo podatkovnih bitov pretvornika in podatkovnega vodila procesorja.



Slika 10. 11: Leva poravnava podatkovnih bitov 12-bitnega pretvornika (d11-d0) v 16-bitni besedi mikroračunalnika (D15-D0)

Naslednja naloga bo prilagajanje digitalnih ekvivalentov merjenega toka normiranim vrednostim, ki jih poznamo iz poglavja 10.3.4.2. Normirani digitalni ekvivalent toku 30 A je

$$\frac{30\,\mathrm{A}}{4{,}8\,\mathrm{A}} \cdot 1000_{\mathrm{HEX}} = 6400_{\mathrm{HEX}}.$$

Če to primerjamo s poravnano vrednostjo A/D pretvornika, ki ustreza toku 30 A ($7ff0_{HEX}$), lahko izračunamo faktor s katerim je treba množiti vhodno vrednost toka za prilagoditev formatu f4.12:

$$\frac{6400_{\mathrm{HEX}}}{7ff0_{\mathrm{HEX}}} = \frac{25600_{\mathrm{DEC}}}{32752_{\mathrm{DEC}}} = "0{,}78163" \rightarrow 0c82_{\mathrm{HEX}}.$$

Poglejmo, ali izračun zares drži? Pri nazivnem toku 4,8 A bomo na A/D pretvorniku dobili vrednost

$$4{,}8\,\mathrm{A} \cdot \frac{7ff_{\mathrm{HEX}}}{30\,\mathrm{A}} = 4{,}8\,\mathrm{A} \cdot \frac{2047_{\mathrm{DEC}}}{30\,\mathrm{A}} = 327{,}5_{\mathrm{DEC}} = 148_{\mathrm{HEX}}$$

oz. po poravnavi v levo 1480_{HEX}. Po množenju z $0c82_{HEX}$ v mikroprocesorju dobimo:

$1480_{HEX} \cdot 0c82_{HEX} = 0100\ 6900_{HEX}$,

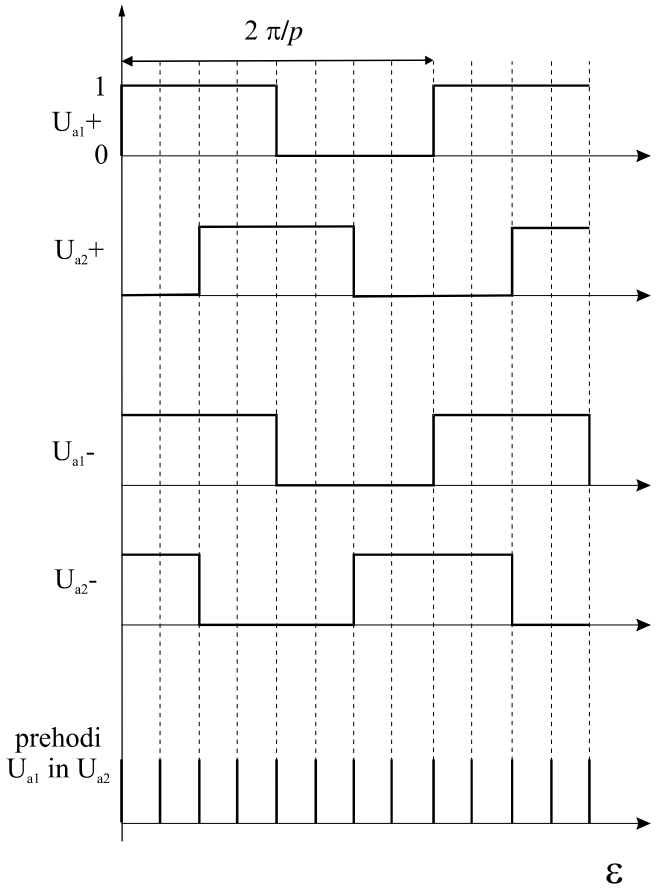
po poravnavi[10] v format f4.12 pa 0100_{HEX}, kar je pričakovani rezultat.

Na podoben problem naletimo pri pošiljanju vrednosti na D/A pretvornik. Postopek je obraten, pri tem pa je treba paziti na format izhodnega podatka, ločljivost pretvornika in zmogljivost močnostnega ojačevalnika.

10.3.4.4.2 Zajemanje kota prek inkrementalnega dajalnika

[10] Štiri bite v levo - upoštevaj MSW ali dvanajst bitov v desno - upoštevaj LSW.

Drugačnego zajemanje in prilagajanje vhodne veličine srečamo pri merjenju kota rotorja. Največkrat izvajamo takšno meritev z inkrementalnim dajalnikom (glej tudi pogl. 12.5.2), ki ob vrtenju generira dva vlaka za 90° premaknjenih pulzov (Ua1 in Ua2 na sliki 10. 12). Trenutni seštevek vseh prehodov obeh vlakov pulzov na digitalnih vhodih (DI na sliki 10. 10) pomeni informacijo o trenutnem kotu.



Slika 10. 12: Oblike pravokotnih signalov iz inkrementalnega dajalnika z ozirom na smer vrtenja (+ ali -)

Predpostavimo, da imamo inkrementalni dajalnik z 2048 pulzi na vrtljaj. Seštevek vseh prehodov obeh vlakov pulzov na en vrtljaj (2π) bo štirikrat tolikšen: $8192_{DEC} = 2000_{HEX}$.

Za štetje so običajno zadolžene posebne enote, ki so sposobne šteti pulze navzgor ali navzdol, odvisno od smeri vrtenja. Štetje je sprotno in neodvisno od programa v CPU-ju, kar omogoča zajemanje zelo hitro spreminjajočega se kota. Enote za zajemanje signalov z dajalnika so bodisi del mikrokontrolerja (npr. TPU pri MC68332 in QEP pri TMS320F240) ali pa ločena vezja (npr. LS2000). Ločljivost števcev je običajno 16 bitov, ponekod pa celo več.

Kot motorja rabimo pri nadaljnjih izračunih v dveh osnovnih blokih (slika 10. 3) za:

1. ugotavljanje hitrosti (odvod kota), kjer rabimo razliko kota med dvema vzorčenjima (glej tudi pogl. 10.3.5.5) ter za

2. transformacijo v koordinatni sistem rotorskega fluksa (blok "ugotavljanje vrednosti fluksa), kjer moramo izračunati sinus in kosinus kota.

Sinus in kosinus sta periodični funkciji, kjer dobimo enako vrednost za kot ε v intervalu [0, 2π], kakor tudi za kot $\varepsilon + n \cdot 2\pi$, kjer je n celo število. Ker sta obe trigonometrični funkciji podani tabelarično (pogl. 10.3.5.2.3), je treba izmerjeni kot (to je trenutni odčitek števca) limitirati na periodo (t.i. *modul 2π*). To dosežemo z enostavnim trikom:
V našem primeru ustreza periodi vrednost 2000_{HEX}. Zato bomo vrednost števca "maskirali" na to območje, npr. z ukazom `AND.W #$2000,D0`, če je podatek shranjen v registru `D0`. Na ta način upoštevamo le del vrednosti števca, ki je pod 2π, vse kar je nad to vrednostjo (mnogokratnik 2π), pa nima pomena. Seveda se problem nekoliko zaplete, če ločljivost dajalnika ni enaka potenci števila dve.

Dobljeni rezultat je načeloma treba še normirati tako, da bo maksimalna vrednost kota 2π ustrezala lokaciji zadnjega podatka v tabeli trigonometrične funkcije (glej tudi pogl. 10.3.5.2.3).

10.3.4.5 Normiranje parametrov

Po normiranju osnovnih veličin (napetost, hitrost, tok pri motorju) sledi še normiranje ostalih parametrov in elektromagnetnih veličin (npr. fluksa, pogl. 10.3.5.1). V tem podpoglavju bomo pokazali princip normiranja samo dveh parametrov motorja: upornosti in induktivnosti. Pri tem izhajamo iz nadomestne enofazne sheme motorja v stacionarnem stanju na sliki 10. 7.

Normiranje upornosti:

Ob osnovni vrednosti toka bo padec napetosti na statorski upornosti:

$$U_{SR} = I_B R_S = 4{,}8 \cdot 3{,}84 \text{ V} = 18{,}432 \text{ V},$$

kar normirano na osnovno vrednost napetosti znaša 18,432/311,3 = "0,0592".

Iz tega sledi, da je normirana upornost tudi "0,0592", digitalni ekvivalent zapisan v formatu f4.12 pa: $0{,}0592 \cdot 4096_{DEC} = 242_{DEC} = f2_{HEX}$.

Normiranje stresane induktivnosti:

Nazivni padec napetosti na stresani induktivnosti $\sigma_S L_0$ ($L_S - L_0 = 0{,}00757$ H, tabela 10.2), to je ob nazivnem toku in frekvenci $\omega_{10} = 2\pi f_{10}$ je določen z izrazom:

$$U_{S\sigma L} = I_B \cdot 2\pi f_{10} \sigma_S L_0 = 4{,}8 \cdot 314{,}159 \cdot 0{,}00757 \text{ V} = 11{,}42 \text{ V},$$

oz. normirano "0,037". Glede na to, da smo pri izračunu upoštevali normirani tok in normirano hitrost ("1"), ustreza izračunana vrednost že stresani induktivnosti. Njen digitalni ekvivalent v formatu f4.12 je $0{,}037 \cdot 4096_{DEC} = 152_{DEC} = 98_{HEX}$.

Poglejmo sedaj, kako bi potekal izračun nazivnega padca napetosti na stresani induktivnosti v računalniku:

1. Množenje ω_{10} in $\sigma_S L_0$ (oba v f4.12): $1000_{HEX} \cdot 98_{HEX} = 98000_{HEX}$,
2. Prilagajanje rezultata množenja[11] formatu f4.12: 98_{HEX},
3. Množenje $\omega_{10} \cdot \sigma_S L_0$ z I_B: $98_{HEX} \cdot 1000_{HEX} = 98000_{HEX}$,
4. Poravnava rezultata (padca napetosti) v f4.12: 98_{HEX}.

10.3.4.6 Alternativni način normiranja in formatiranja

Vse veličine normiramo glede na maksimalne vrednosti (npr. maksimalni tok pretvornika ali motorja, napetost pretvornika in hitrost motorja). Njim dodelimo maksimalne številske vrednosti, ki jih lahko zapišemo s šestnajstimi biti: 8000_{HEX} oz $7fff_{HEX}$. Takšen zapis bi v bistvu ustrezal formatu f1.15. V naslednjem koraku izračunamo digitalne ekvivalente nazivnih vrednosti osnovnih in izvedenih veličin ter parametrov (upornosti, induktivnosti itd.).

Osnovna prednost tega načina je v boljšem izkoristku številskega področja. Po drugi strani pa se bodo digitalni ekvivalenti nazivnih vrednosti posameznih fizikalnih veličin razlikovali.

10.3.5 Realizacija nekaterih splošno uporabnih funkcij

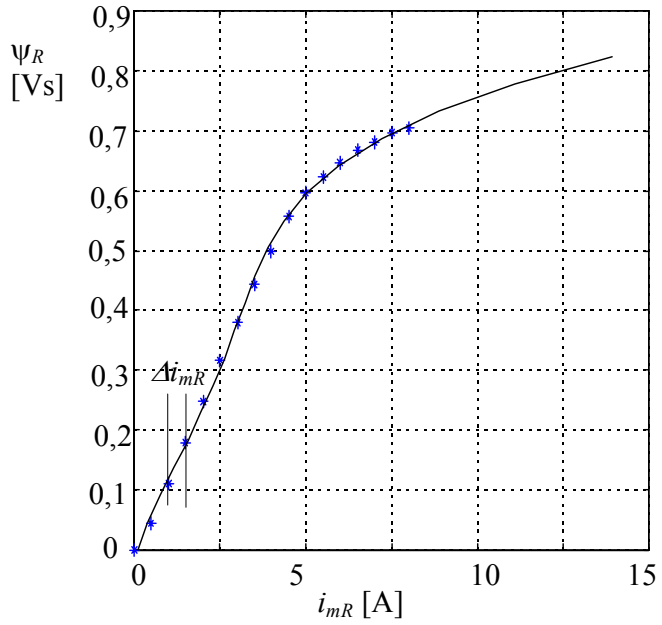
V tem učbeniku si bomo ogledali realizacijo nekaterih zelo pogostih členov in funkcij na procesnem mikroračunalniku ob uporabi celoštevilske aritmetike (zaporedne številke označujejo bloke na slikah 10. 3 in 10. 4):

① nelinearne funkcije,
② trigonometrične funkcije (sinus in kosinus),
③ člen prvega reda,
④ integralni člen,
⑤ diferencialni člen,
⑥ PI regulator,
⑦ pulzno-širinska modulacija.

10.3.5.1 Primer realizacije nelinearne funkcije

V algoritmu pogosto nastopajo zapletene matematične funkcije, ki jih ne moremo realizirati z majhnim številom aritmetičnih ukazov. Takšen primer, ki sicer ni prikazan na prejšnjih blokovnih shemah, je upoštevanje magnetilne krivulje v algoritmu za regulacijo asinhronskega motorja [2]. V prvem delu krivulje (slika 10. 13), je odvisnost izhodne spremenljivke (fluksa Ψ_R) od vhodne veličine (magnetilnega toka i_{mR}) skorajda linearna, dokler ne pride do nasičenja.

[11] Pomik za 12 bitov v desno in upoštevanje le LSW ali pomik za štiri bite v levo in upoštevanje MSW.



Slika 10. 13: Nelinearna funkcijska odvisnost fluksa od magnetilnega toka

Pri reševanju zgornjega problema lahko načeloma izberemo podobni poti, ki sta opisani v poglavju o realizaciji trigonometričnih funkcij.

10.3.5.1.1 Tabeliranje funkcije

V pomnilniško področje shranimo nekaj normiranih vrednosti izhodne spremenljivke (označene z zvezdico na sliki 10. 13)[12]. Postopek vpisovanja in branja tabeliranih vrednosti je enostaven:

1. Z meritvijo smo dobili N parov točk (koordinat) funkcije $\Psi_R = \Psi_R(i_{mR})$, torej bo tabela vsebovala N podatkov. Korak povečanja vhodne spremenljivke, to je razlika med dvema sosednjima podatkoma v tabeli Δi_{mR}, je konstanten. Na k-ti lokaciji, ki ustreza toku $i_{mR}(k)$, se nahaja podatek $\Psi_R(k)$, na naslednji $(k + 1)$ se nahaja podatek $\Psi_R(k + 1)$. Ta ustreza vhodni spremenljivki $i_{mR}(k + 1)$ oziroma $i_{mR}(k) + \Delta i_{mR}$.
2. Pri branju tabeliranih podatkov moramo najprej vhodni podatek $i_{mR}(k)$ zmnožiti s faktorjem $N/i_{mR}(N)$, kjer sta N število tabeliranih podatkov in $i_{mR}(N)$ zadnja zajeta vrednost vhodne spremenljivke v tabeli. Rezultat zaokrožimo na celo vrednost. Tako dobljen zmnožek je v bistvu indeks iskanega podatka v tabeli oz. njegov relativni naslov glede na začetek tabele (angl. offset). Za to lahko uporabimo ukaz za prenos z posrednim naslavljanjem s pomočjo naslovnega registra in odmika (glej tudi pogl. 4.3). Tako bo efektivni naslov podatka enak seštevku naslova začetne lokacije tabele (v odmiku: `zactabele`) in offset-a (v naslovnem registru `A0`):

```
MOVE (zactabele,A0),D0
```

[12] **Pozor**! Vrednosti na sliki ustrezajo realnim vrednostim v koherentnih enotah. Pri manipuliranju s temi vrednostmi v mikroračunalniku jih moramo najprej normirati, kot je to opisano v predhodnih poglavjih.

Vsekakor je zelo malo verjetno, da bodo vhodne vrednosti točno ustrezale tabeliranim abscisnim vrednostim. Zato se je treba odločiti, kateri izmed dveh sosednjih točk je bližja vhodna vrednost in prebrati vsebino njene lokacije. Če menimo, da je napaka prevelika (delitev je pregroba), lahko opravimo linearno interpolacijo (glej naslednje poglavje).

Primer:

Najprej bomo normirali vhodne in izhodne vrednosti s slike 10. 13. Predpostavimo, da ustrezata magnetilnemu toku 4,8 A in pripadajočemu fluksu 0,584 Vs vrednosti 1000_{HEX}, (slika 10. 14). Tabela se začne na naslovu $11f0_{HEX}$.

naslov podatka v tabeli (*HEX*)	*vsebina tabele (normirani* Ψ_R*) v HEX*	i_{mR} [A] *in pripadajoči fluks* [Vs]
1200		4,0 (.....)
1202	f57	4,5 (0,56)
1204	1070	5,0 (0,60)
1206	10fc	5,5 (0,62)
1208	1188	6,0 (0,64)
120a	1215	6,5 (0,66)
120c		7,0 (.....)

Slika 10. 14: Prikaz dela tabele magnetilne krivulje

10.3.5.1.2 Realizacija nelinearne funkcije s pomočjo polinoma

Nelinearno funkcijo v določenem območju lahko aproksimiramo s polinomom višjega reda. Koeficiente polinoma, ki je najboljši približek krivulje skozi izmerjene točke, je možno zelo enostavno izračunati z enim izmed komercialnih programov (npr. Matlab). Stopnja polinoma je odvisna od oblike aproksimirane funkcije, največja omejitev pri izbiri višje stopnje pa je čas potreben, za izračun polinoma v mikroprocesorju. Polinom je treba za vsako vrednost znova izračunati. Polinom četrtega reda, ki ustreza točkam na nenormirani krivulji s slike 10. 13, kaže naslednja enačba:

$$\Psi_R = a_4 i_{mR}^4 + a_3 i_{mR}^3 + a_2 i_{mR}^2 + a_1 i_{mR}^1 + a_0 =$$
$$= 2{,}18 \cdot 10^{-5} \cdot i_{mR}^4 - 8{,}24 \cdot 10^{-4} \cdot i_{mR}^3 + 7{,}71 \cdot 10^{-3} \cdot i_{mR}^2 + 4{,}28 \cdot 10^{-2} \cdot i_{mR}^1 - 7{,}04 \cdot 10^{-4}.$$

Kot vidimo, je za izračun funkcijske vrednosti treba vsakič opraviti veliko operacij seštevanja in množenja (potenciranja in množenja s konstanto), pri čemer sodijo zadnje operacije pri klasičnih mikroprocesorjih med časovno najdaljše. Primernejši od prejšnje oblike je t.i. *Hornerjev zapis*

$$\Psi_R = ((((\underbrace{(a_4 i_{mR} + a_3)}_{1} \cdot i_{mR} + a_2) \cdot i_{mR} + a_1) \cdot i_{mR} + a_0),$$

ki ga enostavno izvedemo na DSP-jih. Ti vsebujejo ukaz za množenje s hkratnim prištevanjem k prejšnjemu rezultatu. Zgornji polinom bi tako lahko izračunali s štirimi ukazi. Nikakor pa ne gre pozabiti, da mora vsak algoritem poleg zgoraj navedenih matematičnih operacij vsebovati še dodatne operacije, ki preprečujejo nezaželena stanja (npr. preverjanja in omejitve zaradi morebitnega presežka v registru itd.). Zato so programerji ta način podajanja funkcije v časovno kritičnih sistemih uporabljali le izjemoma.

10.3.5.2 Primer realizacije trigonometričnih funkcij

Trigonometrične funkcije so zelo pogost element v regulacijskih shemah, ki opisujejo dogajanja v električnih sistemih, kot so npr. omrežno napajana bremena ali regulirani izmenični pogoni. V blokovnih shemah iz našega zgleda jih srečujemo pri transformacijah veličin iz enega pravokotnega koordinatnega sistema v drugi, katerega abscisa je premaknjena za nek kot ρ. Takšen blok je na sliki 10. 4 označen z ②.

10.3.5.2.1 Uporaba standardnih trigonometričnih funkcij

Pri realizaciji katerekoli funkcije v realnem času v sistemih z zelo majhnimi časovnimi konstantami poskušamo trajanje algoritma zmanjšati na najkrajši možni čas, zato uporaba trigonometričnih funkcij v višjih jezikih (npr. C, Pascal, Basic...) v našem primeru ne pride v poštev. Operacije so del že obstoječih knjižnic programa (npr. `sin(a)` v C jeziku). Čeprav je uporaba teh funkcij elegantna, pa časovno ni optimalna, kar je splošno znan problem višjih jezikov. Le v primeru, ko imamo opravka z izredno hitrimi procesorji (npr. DSP-ji) ali počasnimi procesi, si lahko dovolimo njihovo uporabo. Največkrat mora programer funkcijo, ki ustreza tipičnim zahtevam njegovega progama, zelo skrbno programirati v zbirnem jeziku. Tudi najsodobnejši prevajalniki dosežejo le (ali že) 80 % časovnega optimiranja, ki ga doseže izkušeni programer.

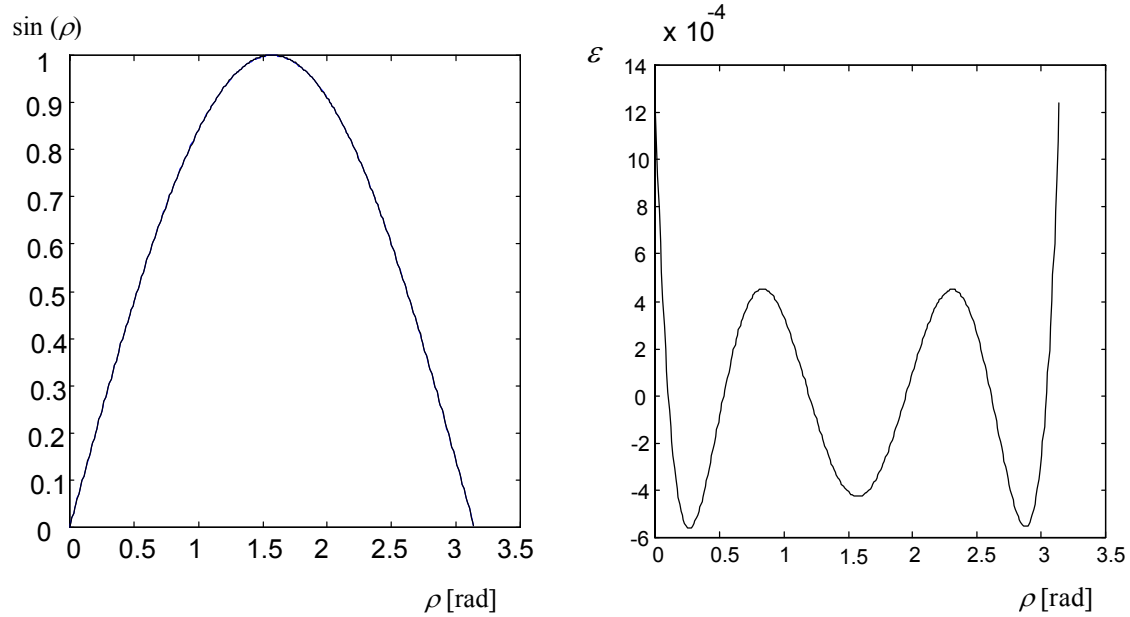
10.3.5.2.2 Zapis sinusne funkcije v obliki polinoma

Vsako trigonometrično funkcijo lahko razvijemo v neskončno vrsto, večinoma pa zadostuje že nekaj prvih členov. Ta način zapisa ni optimalen na DSP-ju, ker vsebuje le sode ali lihe potence. Zato DSP ukaza za množenje in hkratno seštevanje iz poglavja 10.3.5.1.2 ne moremo izkoristiti v največji možni meri.

Če se že odločimo za vsakokratni izračun trigonometrične funkcije, raje uporabimo zapis v obliki polinoma. Npr. vsako od polperiod sinusne funkcije izračunamo s posebnim polinomom:

$$\sin\rho = \begin{cases} \Big(\Big(\Big((0{,}0372\cdot\rho) - 0{,}2338\Big)\cdot\rho + 0{,}0544\Big)\cdot\rho + 0{,}9826\Big)\cdot\rho + 0{,}0013, & \text{če } \rho \geq 0 \\ \Big(\Big(\Big((0{,}0372\cdot\rho) + 0{,}2338\Big)\cdot\rho + 0{,}0544\Big)\cdot\rho - 0{,}9826\Big)\cdot\rho + 0{,}0013, & \text{če } \rho < 0 \end{cases}$$

Slika 10. 15 kaže sinusno funkcijo, ki smo jo izračunali s pomočjo zgornje vrste ter razliko (pogrešek) med tako dobljenimi vrednostmi funkcije in pravo vrednostjo sinusa v odvisnosti od kota.



Slika 10. 15: Potek sinusne funkcije, ki je bila izračunana s pomočjo končne vrste in odstopanje od tiste neskončne vrste

10.3.5.2.3 Tabelarično podajanje trigonometričnih funkcij

Najpogosteje uporabljan način izračuna trigonometričnih funkcij v časovno zahtevnih sistemih je metoda tabelaričnega podajanja vrednosti, podobno kot pri nelinearnih funkcijah iz pogl. 10.3.5.1.1. Velja omeniti nekaj koristnih napotkov:

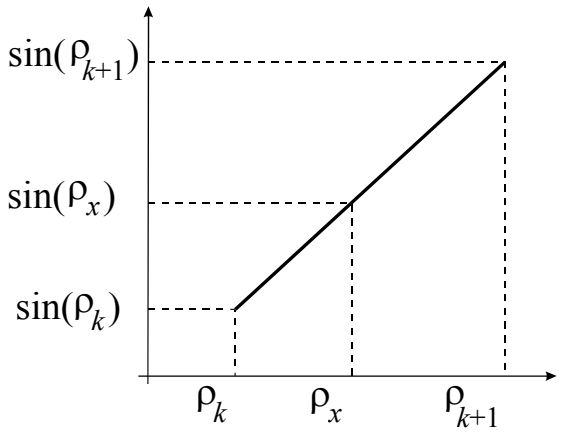
Število tabeliranih vrednosti:

Pričakovati je, da je natančnost podajanja funkcije s pomočjo tabele odvisna od števila elementov tabele (ločljivost). Po drugi strani bo preveliko število elementov zasedlo preveč prostora v pomnilniku. Kompromis lahko dosežemo tako, da tabeliramo le vrednosti sinusa, ki ustrezajo četrtini periode, saj so vrednosti v ostalih treh kvadrantih zrcalne podobe prvega kvadranta s pozitivnim ali z negativnim predznakom (drugi in četrti kvadrant) oziroma kar negativne vrednosti funkcije za prvi kvadrant (tretji kvadrant). Seveda bo celoten algoritem, ki ugotavlja kvadrant podanega kota in relacijo s tabeliranim kotom v prvem kvadrantu, nekoliko daljši.

Število podatkov nadalje zmanjšamo (večji korak med dvema vrednostma) tako, da kote, ki niso zajeti v tabeli, izračunamo z linearno interpolacijo (10. 3)

$$\sin(\rho_x) = \sin(\rho_k) + \frac{\rho_x - \rho_k}{\rho_{k+1} - \rho_k} \cdot \left(\sin(\rho_{k+1}) - \sin(\rho_k)\right), \qquad (10.\ 3)$$

kjer sta s k in k+1 označeni dve zaporedni tabelirani vrednosti, ρ_x pa je kot, katerega sinus želimo izračunati (slika 10. 16).



Slika 10. 16: Linearna interpolacija za vrednosti med dvema podatkoma v tabeli

Število podatkov v tabeli (ločljivost) je med ostalim odvisno od načina normiranja, navadno pa je enako n-ti potenci števila 2, npr. $2^8 = 256$, $2^9 = 512$, $2^{10} = 1024$.

Normiranje kota in vrednosti sinusa:

Normiranje kota:

Vrednosti kotov in pripadajočih sinusov so normirane (glej podpoglavja o zajemanju kota rotorja 10.3.4.4.2 in integratorju 10.3.5.4). Koti imajo le pozitivne vrednosti (0 - 2π), zato logike dvojiškega komplementa pri normiranju zaradi boljše ločljivosti načeloma ne upoštevamo. Tako bodo normirane vrednosti **nepredznačene**, kar pri 16-bitnem normiranju pomeni vrednosti od 0000_{HEX} do $ffff_{HEX}$ (format f0.16). Ena od prednosti takega formatiranja se kaže v samodejnem izvajanju operacije "modul 2π": ob povečanju zadnje nepredznačene vrednosti $ffff_{HEX}$ (kot 2π/65535) bo naslednja vrednost 0_{HEX} (kot 0).

Zaradi poenostavitve bomo v nadaljevanju predpostavili, da tabeliramo sinusne vrednosti za celotno periodo (0 - 2π) in ne le četrtino. V tabelo seveda ne bomo vpisali sinusnih vrednosti za vseh 65536 vrednosti kotov, ampak izberemo bistveno manjše število, npr. eno izmed predlaganih v prejšnji točki, ki je potenca števila 2. Prednost tega pristopa je v enostavni prilagoditvi kota dolžini tabele s pomočjo operacije pomika ("shift") v desno (deljenje z 2^n). Pri izbiri $2^8 = 256$ podatkov (od 0 do 2π) moramo pred branjem tabele normirano vrednost kota premakniti za osem bitov v desno ($2^{16}/2^8 = 2^8$), pri $2^9 = 512$ podatkih za sedem bitov ($2^{16}/2^7 = 2^9$) itd.
V našem zgledu pa bomo zaradi lažje obravnave in kompatibilnosti s formati ostalih veličin kote normirali v formatu f4.12, pri čemer bo kotu 2π ustrezala vrednost "1" ali 1000_{HEX}. Štirje

zgornji biti bodo vedno nič, kar dosežemo z maskiranjem v modul 2π (glej pogl. 10.3.4.4.2)[13]. Tej operaciji sledi še prilagoditev območja kota dolžini tabele, kar dosežemo na enak način, kot smo to naredili prej: s premikom vrednosti kota za določeno število bitov v desno, odvisno od dolžine tabele.

Normiranje vrednosti sinusa:

Zaloga vrednosti sinusa ali kosinusa je med -1 in +1 (normirano "-1" do "+1"), zato načeloma tabelirane vrednosti določimo z upoštevanjem dvojiškega komplementa (8000_{HEX} = -32768_{DEC} do $7FFF_{HEX}$ = 32767_{DEC} pri 16-bitnem normiranju). Tako v celoti izkoristimo 16-bitni obseg.

Če pa želimo obdržati format f4.12, bo normirani vrednosti "-1" ustrezal podatek $f000_{HEX}$, "+1" pa 1000_{HEX}.

Primer:

V tabelo smo shranili 512 (2^9)vrednosti celotne periode sinusne funkcije. Na začetni, "ničti" lokaciji, ki ustreza kotu 0·2π/512 se nahaja vrednost sinusne funkcije za ta kot (0), na naslednji lokaciji (kot 1·2π/512) se nahaja vrednost sin(0,01227) = 0,0123 ⇔ "0,0123" · 4096_{HEX} = 50_{DEC} = 32_{HEX}. Tretja lokacija (kot 2·2π/512) vsebuje vrednost 65_{HEX} ("0,0245" · 4096_{DEC} = 101_{DEC}), zaradi sin(0,02456) = 0,02455. Maksimalna pozitivna vrednost na 127. lokaciji (kot 127·2π/512 ⇔ π/2) pa je enaka 1000_{HEX}[14]. Maksimalna negativna vrednost sinusne funkcije se nahaja na 384. lokaciji in je enaka $f000_{HEX}$ ⇔"-1". Nekatere vrednosti kaže tabela 10.4. Zaradi omejene ločljivosti so vrednosti v bližini amplitude, kjer je sprememba najmanjša, enake.

Naslov podatka (**Začetni naslov tabele +**)	Kot (rad)	Vsebina lokacije (HEX)	Vsebina lokacije (DEC)	Vrednost sinusa
0	0	0	0	0
1	0,01223	32	50	0,0122
2	0,0245	65	101	0,0245
...				
127_{DEC} = $7f_{HEX}$	1,559	1000	4096	0,9999
...				
383_{DEC} = $17f_{HEX}$	4,7	f000	-4096	-0,9999
384_{DEC} = 180_{HEX}	4,7123	f000	-4096	-1
385_{DEC} = 181_{HEX}	4,7247	f000	-4096	-0,9999
...				
511_{DEC} = $1ff_{HEX}$	6,27	ffce	-50	-0,0123

Tabela 10.4: Nekatere vrednosti tabelirane funkcije sinusa v formatu f4.12

[13] Če imamo opravka z inkrementalnim dajalnikom z 8192_{DEC} = 2000_{HEX} pulzi (4·2048; pogl. 10.3.4.4.2), dosežemo pretvorbo v format f4.12 (po maskiranju v modul 2π) s premikom za en bit v desno.

[14] Ponovno se moramo spomniti, da je pri dvojiškem komplementu negativni obseg za eno vrednost večji od pozitivnega.

10.3.5.2.4 Kratka primerjava opisanih načinov izračuna trigonometrijskih funkcij

Pri klasičnih mikroprocesorjih ima tabelarično podajanje funkcij prednost pred razvojem v vrsto zaradi manjšega števila množenj, ki zahtevajo bistveno več procesorskega časa. Pri DSP-jih pa je stanje popolnoma drugačno: operacije množenja zahtevajo približno enako število ukaznih ciklov kot ostale operacije. Zato je pri njih čas izvajanja algoritma za izračun tabelarično podane trigonometrične funkcije daljši od časa, ki je potreben za razvoj v vrsto na DSP TMS320F240 (tabela 10.5) [28].

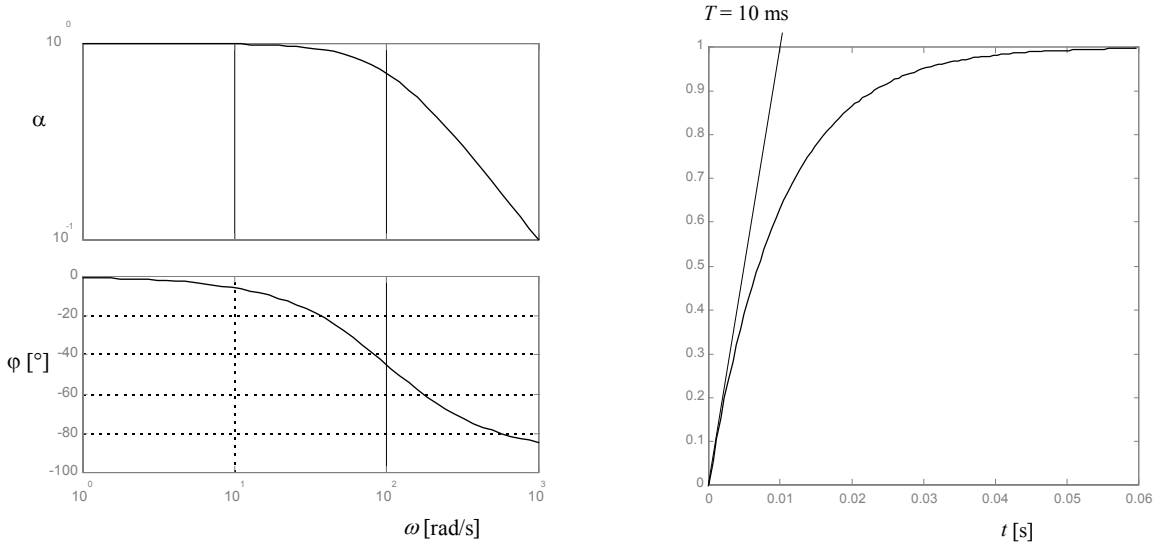
	Čas izvajanja funkcije (v μs)	
Funkcija	Tabelarično podajanje in interpolacija	Razvoj v vrsto
sin	1,75	1,35
cos	1,90	1,85

Tabela 10.5: Primerjava časa izvajanja trigonometričnih funkcij v zbirnem jeziku s pomočjo razvoja v vrsto in tabele (DSP)

10.3.5.3 Primer realizacije člena prvega reda

10.3.5.3.1 Opis lastnosti člena prvega reda

Člen prvega reda (angl. first order lag) pogosto srečamo v regulacijskem programu, ki ga je treba izvajati v realnem času (blok ③ na sliki 10. 4) [2,4]. Iz njegove frekvenčne karakteristike (slika 10. 17) sta razvidna amplitudni in fazni potek v odvisnosti od frekvence vhodnega signala. Drugi del slike kaže časovni odziv člena prvega reda na enotino funkcijo (t.i. *prehodna funkcija*). Ob faktorju jačanja $K = 1$ (glej tudi (10. 4)) se člen lahko uporablja kot t.i. *nizkopasovni filter*, saj v področju višjih frekvenc (nad frekvenco $\omega_0 = T^{-1}$), občutno duši amplitudo vhodnega signala.



Slika 10. 17: Frekvenčna karakteristika in prehodna funkcija člena prvega reda (K = 1, T = 10 ms oz. ω_0 = 100 rad/s)

Člen prvega reda opisuje diferencialna enačba prvega reda

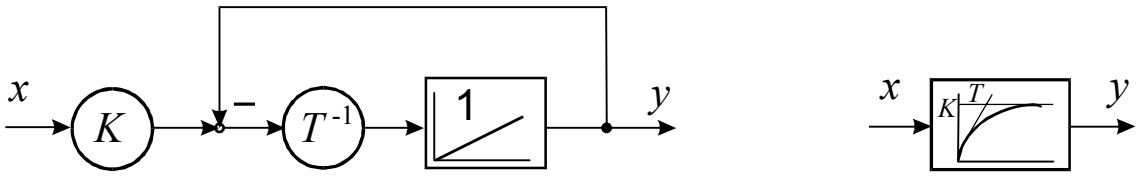
$$T\frac{dy(t)}{dt} + y(t) = K\,x(t)\,, \tag{10. 4}$$

kjer je x vhodna, y pa izhodna veličina ter K ojačenje in T časovna konstanta sistema.

Prenosna funkcija člena prvega reda (razmerje med izhodnim in vhodnim signalom v frekvenčnem prostoru) je določena z izrazom

$$F(p) = \frac{y(p)}{x(p)} = \frac{K}{1+pT} = \alpha\, e^{j\varphi}\,. \tag{10. 5}$$

Na sliki 10. 18 sta možna načina grafičnega prikaza bloka člena prvega reda.



Slika 10. 18: Grafična prikaza člena prvega reda

10.3.5.3.2 Diferenčna enačba člena prvega reda

Člen prvega reda v mikroračunalniku realiziramo z numeričnim reševanjem diferencialne enačbe (10. 4). Glede na diskretno naravo delovanja procesnega mikroračunalnika, ki ciklično, s periodo vzorčenja T_V, izvaja regulacijski algoritem, moramo diferencialno enačbo najprej diskretizirati. Na ta način dobimo t.i. *diferenčno enačbo*. Obstaja več različnih pristopov k reševanju diferenčnih enačb. V časovno kritičnih regulacijskih sistemih se verjetno najbolj splača najhitrejša metoda, ki ni nujno tudi najbolj natančna.

Diferenčno inačico enačbe (10. 4) dobimo tako, da izračunamo odvod v končno kratkem intervalu, to je času vzorčenja T_V. Velja:

$$\lim_{\Delta t\to 0}\frac{\Delta y}{\Delta t} = \lim_{T_V\to 0}\frac{\Delta y}{T_V} = \lim_{T_V\to 0}\frac{y(k)-y(k-1)}{T_V} = \frac{dy}{dt}\,, \tag{10. 6}$$

kjer je Δy razlika (*diferenca*) vrednosti izhodne spremenljivke v trenutnem (k) in prejšnjem vzorcu (k-1), ob zelo kratkem intervalu vzorčenja T_V. Diferenčna enačba člena prvega reda je potemtakem:

$$\frac{T}{T_V}\big(y(k)-y(k-1)\big)+y(k)=K\,x(k). \qquad (10.\ 7)$$

Že iz (10. 6) je razvidna zahteva po čim krajšem času vzorčenja. Ta pa je po drugi strani omejen s trajanjem obdelave celotnega regulacijskega programa, ki ga je treba izvršiti v tem intervalu, ter hitrostjo procesorja. V praksi je velikost T_V odvisen tudi od časovnih konstant sistema (v našem primeru T), saj je pri zelo velikih časovnih konstantah (npr. nekaj minut) nesmiselno vztrajati pri zelo majhnih intervalih vzorčenja (npr. nekaj ms). Običajno je čas vzorčenja vsaj desetkrat (po možnosti tudi nekaj desetkrat) krajši od časovne konstante člena, npr. $T_V = 1$ ms pri $T = 40$ ms.

Rešitev diferenčne enačbe (10. 7) lahko zapišemo v obliki

$$y(k)=\frac{T_V K}{T+T_V}x(k)+\frac{T}{T+T_V}y(k-1) \qquad (10.\ 8)$$

To je *rekurzivna* enačba, v kateri je vrednost izhoda v tekočem vzorcu $y(k)$ odvisna od tekoče vrednosti vhoda $x(k)$ ter vrednosti izhoda $y(k\text{-}1)$, ki smo jo že izračunali v prejšnjem vzorcu. Začetni pogoj za $y(0)$ je običajno 0. Diferenčno enačbo lahko zapišemo tudi v diskretnem z prostoru:

$$F(z)=\frac{bz}{z-a}, \qquad (10.\ 9)$$

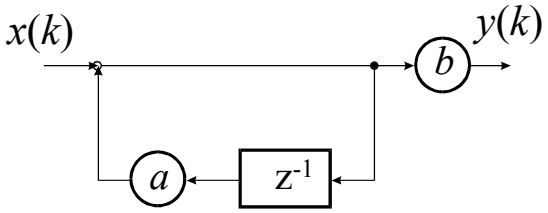
kjer sta v našem primeru

$$a=\frac{T}{T+T_V} \quad \text{in} \quad b=\frac{T_V K}{T+T_V}. \qquad (10.\ 10)$$

Pri tem je z^{-1} je *operator zakasnitve* za eno periodo vzorčenja; glej tudi [2, 13]:

$y(k\text{-}1) = z^{-1}\, y(k)$.

Iz tega sledi tudi blokovna shema s slike 10. 19.



Slika 10. 19: Blokovna shema diskretnega člena prvega reda

10.3.5.3.3 Praktična realizacija člena prvega reda v procesnem mikroračunalniku

Kot primer vzemimo realizacijo člena prvega reda s parametroma $K = 1$, $T = 50$ ms, ki ga bomo realizirali znotraj intervala vzorčenja $T_V = 1$ ms. Enačba 10. 8 se sedaj glasi:

$$y(k) = "0{,}0196" \cdot x(k) + "0{,}9804" \cdot y(k-1).$$

Konstante bomo zaradi združljivosti z ostalimi veličinami in enostavnosti zapisali v formatu f4.12, čeprav bi bil v tem primeru primernejši format f1.15, s katerim bi polno izkoristili ločljivost mikroprocesorja. Tako je prva konstanta 50_{HEX}, druga pa $fb0_{HEX}$.

Prekinitven podprogram za realizacijo člena prvega reda v MC68332 je zelo kratek:

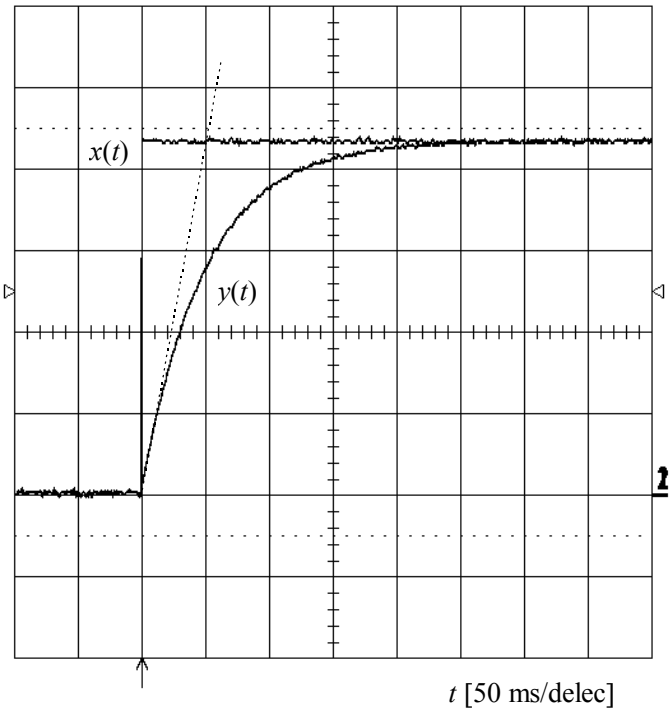
```
*************************************************************
***                                                       ***
***                  CLEN PRVEGA REDA                     ***
***                                                       ***
***       vhod na naslovu (x), izhod na naslovu (y)       ***
***       konstante k1=T/(T+Tv), k2=K*Tv/(T+Tv),          ***
***                                                       ***
*************************************************************

     clr.l     d3              *** brisi registra
     clr.l     d1              *** za vsak primer

     move.w    (x),d3          *** vhod v pt1 x(k)***
     move.w    (y),d1          *** stara vrednost izhoda y(k)

     muls.w    #k1,d1          *** mnozenje s k1=T/(T+Tv)
     muls.w    #k2,d3          *** mnozenje s k2=KTv/(T+Tv)

     add.l     d1,d3           *** y(k)=k2*x(k)+k1*y(k-1)

     asl.l     #4,d3           *** pomik za 4 bite v levo
                               *** (rezultat v zgornji besedi)
     swap.w    d3              *** postavi ga v spodnjo besedo
     move.w    d3,(y)          *** shrani y(k)
```

Pred tem smo določili konstante ter rezervirali pomnilniški prostor za spremenljivke (pogl. 9.1.2):

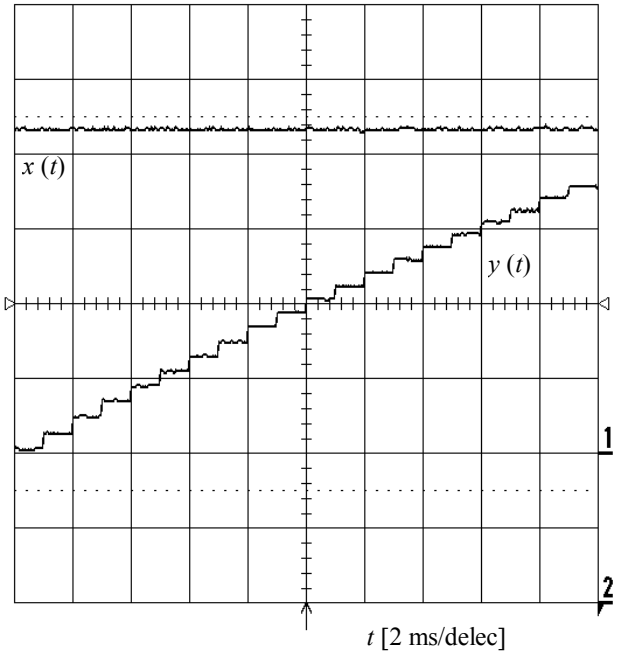
```
k1   equ  $fb0                *** definicija konstante k1
k2   equ  $50                 *** definicija konstante k2

***  rezervacija pomnilniškega prostora

x    ds.w 1                   *** rezerviraj prostor v dolzini
y    ds.w 1                   *** ene besede za x in y
```

Zgornje spremenljivke moramo v fazi inicializacije postaviti v stanje 0.

Slika 10. 20 kaže oscilogram prehodne funkcije člena prvega. Na sliki 10. 21 je prikazan isti odziv ob zmanjšani časovni bazi, kjer se razločno vidi amplitudna in časovna diskretizacija.



Slika 10. 20: Oscilogram prehodne funkcije člena prvega reda



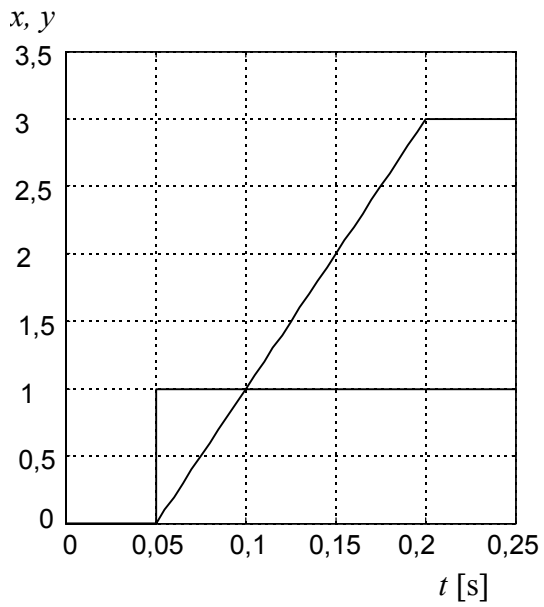
Slika 10. 21: Povečani oscilogram s prejšnje slike

10.3.5.4 Primer realizacije integralnega člena

Integrator je tudi pogost element v regulacijski shemi (npr. blok ⑤ na sliki 10. 4). Pri razvoju ustreznega podprograma izhajamo iz enačbe v časovnem prostoru

$$y(t) = \frac{1}{T_i}\int_0^t x(\tau)d\tau = K_i\int_0^t x(\tau)d\tau\,, \qquad (10.\ 11)$$

kjer je T_i časovna konstanta integralnega člena oz. K_i njena obratna vrednost. Izvajanje integratorskega podprograma mikroprocesorja se začne v trenutku t = 0 s. Časovna konstanta je čas, v katerem doseže izhodna veličina spremembo v velikosti svoje osnovne vrednosti, če deluje na vhodu veličina s konstantno osnovno vrednostjo (slika 10. 22) [4].



Slika 10. 22: Prehodna funkcija integratorja s T_i = 50 ms in vhodna enotina stopnica

Čeprav obstaja več načinov pretvorbe zgornje enačbe v diferenčno obliko, bomo tukaj uporabili najenostavnejšo (Eulerjevo ali pravokotno metodo). Pri končno majhnem času T_V se enačba (10. 11) glasi:

$$y(k) = K_i\sum_{i=0}^{k} x(i)\,T_V = K\sum_{i=0}^{k} x(i)\,, \qquad (10.\ 12)$$

kjer je $K = K_i \cdot T_V$.

Enačbo lahko zapišemo tudi v rekurzivni obliki, saj velja

$$y(k) = Kx(k) + K\sum_{i=0}^{k-1} x(i) = Kx(k) + y(k-1)\,.$$

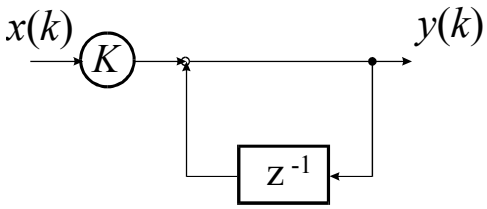
Ob uporabi operatorja zakasnitve, lahko zapišemo

$$y(k) = Kx(k) + z^{-1}y(k),$$

iz tega sledita prenosna funkcija

$$F(z) = \frac{K \cdot z}{z - 1}$$

ter blokovna shema na sliki 10. 23.



Slika 10. 23: Blokovna shema diskretnega integralnega člena

V mikroprocesorju integralni člen realiziramo zelo enostavno. Pri tem je najpomembnejša konstanta *K*, ki je odvisna od integracijske časovne konstante in časa vzorčenja.

Primer:

V mikroprocesorju želimo realizirati integrator s časovno konstanto 50 ms pri T_V = 1 ms. Iz prej povedanega sledi, da mora pri vzbujanju z enotino stopnico (1000_{HEX} pri f4.12) izhod doseči vhodno vrednost v času *t* = 50ms, to je po petdesetih vzorcih. Konstanta *K* iz (10. 12) je torej $4096/50 \approx 82_{DEC} = 52_{HEX}$.

Pri realizaciji integratorja moramo biti še posebej pozorni na omejevanje izhodne veličine (slika 10. 22). Izhodna vrednost integratorja namreč vseskozi narašča (ali upada), dokler ne pride do spremembe predznaka vhoda (izhodna vrednost začne upadati) ali dokler le-ta ne pade na vrednost nič (izhodna vrednost se ustali). Pri relativno kratkih časovnih konstantah integratorja obstaja realna nevarnost, da integrator doseže skrajno mejo možnega zapisa v registru.

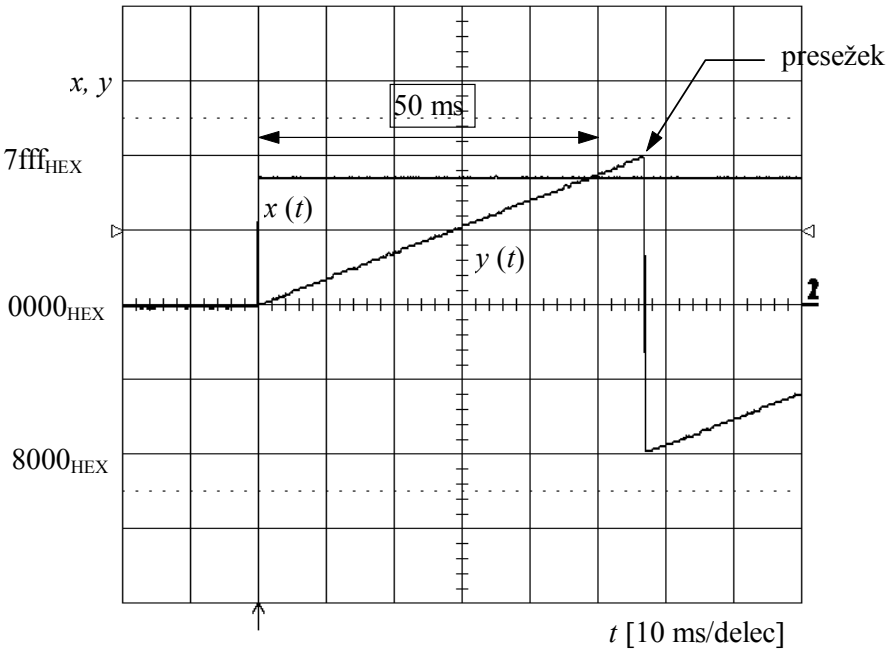
Sledi program za realizacijo integralnega člena iz zgornjega primera v MC68332 in oscilogram odziva (brez inicializacijskega dela):

```
*************************************************************
***                                                       ***
***                  INTEGRALNI CLEN                      ***
***                                                       ***
***       vhod na naslovu (x), izhod na naslovu (y)       ***
***       vmesni integral (yvmes) v formatu f8.24         ***
***                  konstanta ojacenja: k                ***
***                                                       ***
*************************************************************

     clr.l     d1              *** brisi registra
     clr.l     d0              *** za vsak primer

     move.w    (x),d1
     muls.w    #k,d1           *** mnozi s konstanto k = 82

     move.l    (yvmes),d0
     add.l     d1,d0           *** yvmes = yvmes + x*k
     move.l    d0,(yvmes)      ***   PAZI!   Vmesna   vrednost
(yvmes)
                               *** se vedno ni premaknjena za
12
                               *** bitov v desno oz. 4 v levo!

     asl.l     #4,d0           ***  sele  sedaj  poravnava  v
f4.12.
                               *** Rezultat v zgornji besedi!

     swap.w    d0              *** postavi ga v spodnjo besedo

     move.w    d0,(y)          *** Dejanski izhod v f4.12
```

V programu razlikujemo dve vrednosti integrala. Prva (`yvmes`) je seštevek produktov s konstanto v 32-bitnem formatu f8.24 (f4.12 ·f4.12 = f8.24). Na ta način zmanjšamo pogrešek pri kontinuiranem seštevanju, saj računamo vseskozi z 32-bitno ločljivostjo. Šele pred prenosom na lokacijo dejanske izhodne vrednosti (`y`) opravimo poravnavo za 4 bite v levo ter upoštevamo besedo z najmanjšo težo. Zaradi tega moramo za spremenljivko `yvmes` v RAM-u rezervirati prostor 4 byte:

```
yvmes     ds.l      1
```

V zgledu smo namenoma spustili del za omejevanje integratorja zaradi ilustracije možnih negativnih posledic. Slika 10. 24 kaže rezultat algoritma.



Slika 10. 24: Oscilogram prehodne funkcije integratorja v realnem času

Izhodna veličina doseže vrednost vhodne veličine (tukaj 7000_{HEX}) po času T_V = 50 ms ter nadaljuje z naraščanjem, ker je vhodna veličina še vedno pozitivna. Ko doseže maksimalno pozitivno vrednost 16-bitnega registra ($7fff_{HEX}$), pride do presežka, zato izhodna vrednost integratorja nadaljuje svoj potek z negativno vrednostjo (8000_{HEX}). To ne ustreza realnemu integratorju in pripelje do nepredvidljivega obnašanja celotne regulacijske proge!

10.3.5.5 Primer realizacije diferencialnega člena

Diferencialni (D) člen (npr. ⑥ na sliki 10. 3) deluje kot ojačevalnik visokofrekvenčnih motenj, zato je eden najmanj zaželenih elementov vsakega regulacijskega sistema. To še posebej velja za diskretne sisteme [13]. V primerih, ko je diferencialni člen neizogiben (npr. za določanje kotne hitrosti rotorja iz merjenega kota $\omega = \mathrm{d}\varepsilon/\mathrm{d}t$), mu navadno dodamo še nizkopasovni filter. Moramo se zavedati, da s tem namenoma poslabšamo dinamično karakteristiko diferencialnega člena, ki je ojačevalnik spremembe vhodne veličine, torej zaznava tendenco poteka. Zato je izbira časovne konstante filtra (konstanti a in b v (10. 9) oz. (10. 10)) kompromis med motilno frekvenco in pričakovano dinamiko sistema.

Realizacija enostavnega diferencialnega člena sledi iz (10. 6):

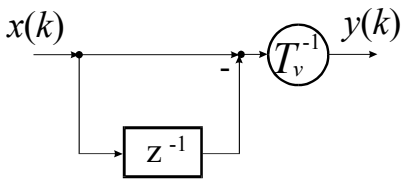
$$\frac{dy}{dt} \approx \frac{y(k) - y(k-1)}{T_V}$$

Prenosno funkcijo dobimo z uvedbo operatorja zakasnitve:

$$\frac{y(z) - z^{-1}y(z)}{T_V} = x(z) \quad \Rightarrow$$

$$F(z)=\frac{y(z)}{x(z)}=\frac{z-1}{T_V z} \qquad (10.\ 13)$$

Blokovna shema je prikazana na sliki 10. 25.

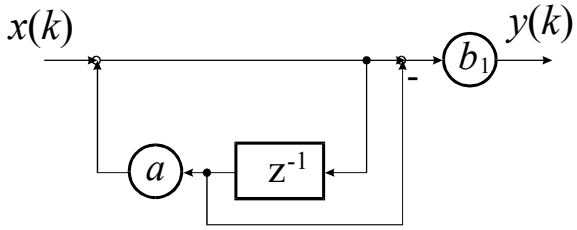


Slika 10. 25: Blokovna shema diskretnega diferencialnega člena

Povedali smo že, da pri praktični realizaciji D členu običajno dodamo še nizkopasovni filter. Iz (10. 9) in (10. 13) sledita prenosna funkcija tega člena in blokovna shema.

$$F(z)=\frac{bz}{z-a}\cdot\frac{z-1}{T_V z}=\frac{b}{T_V}\frac{z-1}{z-a}=b_1\frac{z-1}{z-a},$$

kjer je $b_1 = b/T_V = K/(T+T_V)$ (glej tudi(10. 10)).



Slika 10. 26: Prenosna funkcija filtriranega diferencialnega člena

Primer:

Opisali bomo enostavno realizacijo odvajanja kota, ki smo ga zajeli z inkrementalnim dajalnikom (ⓒ na sliki 10. 3 in pogl. 10.3.4.4.2) in hkratno prilagajanje normiranim vrednostim. Nizkopasovni filter v tem prikazu ni zajet.

Polni vrtljaj rotorja ustreza 8192_{DEC} (2000_{HEX}) pulzom dajalnika. Čas vzorčenja hitrosti (superponirana regulacijska zanka) je daljši od osnovnega intervala vzorčenja (npr. 100 μs), saj so tudi mehanske konstante bistveno večje of električnih. V našem primeru bomo upoštevali $T_V = 3$ ms. Nazivna hitrost 148,7 rad^{-1} (ali 1420 min^{-1}; tabela 10.2) je normirana na nazivno osnovno frekvenco statorskega navitja (50 Hz ali 314,159 rad/s), ki ji ustreza digitalna vrednost 1000_{HEX} v formatu f4.12. Torej ustreza nazivni hitrosti vrednost "0,473" ali $1939_{DEC} = 793_{HEX}$. V eni sekundi se pri nazivni hitrosti rotor zavrti 1420/60 = 23,667-krat, v enem intervalu vzorčenja kota pa naredi 23,667·0,003 = 0,071 vrtljaljev. To ustreza $0{,}071 \cdot 8192 = 582_{DEC} = 246_{HEX}$ pulzom inkrementalnega dajalnika (Δy iz diferenčne enačbe

(10. 6)). Torej bo pri normirani nazivni hitrosti (1939_{DEC} = 793_{HEX}) razlika preštetih pulzov inkrementalnega dajalnika med dvema vzorčenjima enaka 582_{DEC}. Iz tega sledi, da bomo za meritev poljubne hitrosti pomnožili število preštetih pulzov s faktorjem $1939_{DEC}/582_{DEC}$ = "3,3316", kar v formatu f4.12 pomeni množenje s številom 13646_{DEC} = $354e_{HEX}$!

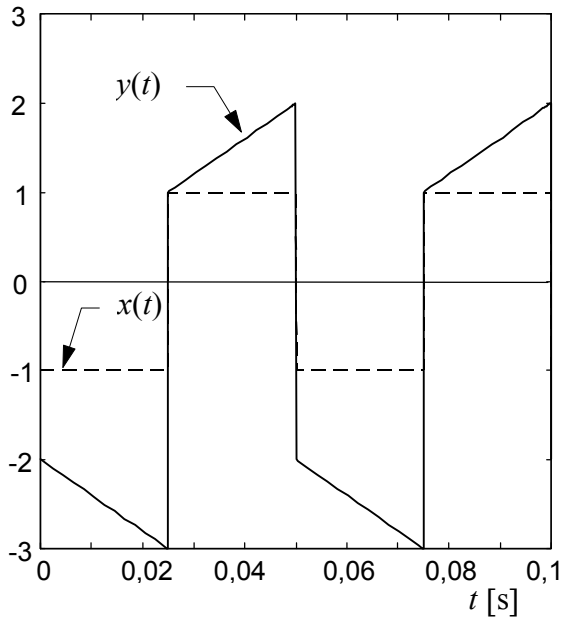
10.3.5.6 Primer realizacije PI člena

PI (⑦) je tip regulatorja, ki ga najpogosteje srečujemo v zahtevnejših elektromehanskih sistemih. Gre v bistvu za hkratno delovanje (seštevek) integralnega in proporcionalnega člena. S tem dosežemo dobro dinamiko in odpravimo statični pogrešek. Prenosna funkcija PI regulatorja v Laplaceovem prostoru je [4]:

$$F(p) = K_p + \frac{1}{T_i p} = \frac{K_p T_i p + 1}{T_i p} = K_p \frac{T_{ip} p + 1}{T_{ip} p}, \qquad (10.\ 14)$$

kjer je $T_{ip} = K_p T_i$.

Slika 10. 27 kaže odziv PI člena na simetričen vlak pulzov. Ojačenje proporcionalnega člena je 2, časovna konstanta integratorja T_i pa 25 ms, začetna vrednost vhodnega signala je enaka nič.

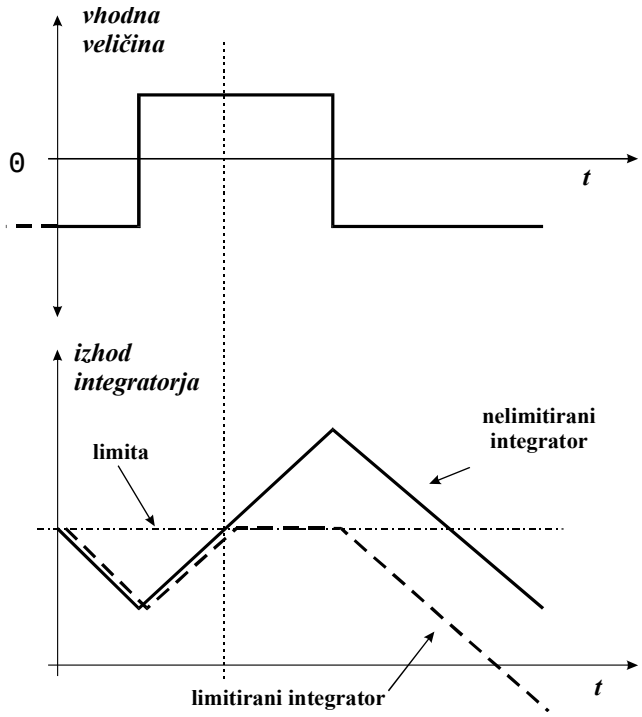


Slika 10. 27: Odziv PI člena na simetričen vlak pulzov (normirano)

Glavni problem pri programiranju PI člena je usklajevanje limit. Do presežka področja registra pride bodisi v posameznih vejah PI člena (v proporcionalnem delu pri velikem ojačenju ali veliki vrednosti vhoda; v integralnem delu pri majhnih časovnih konstantah,

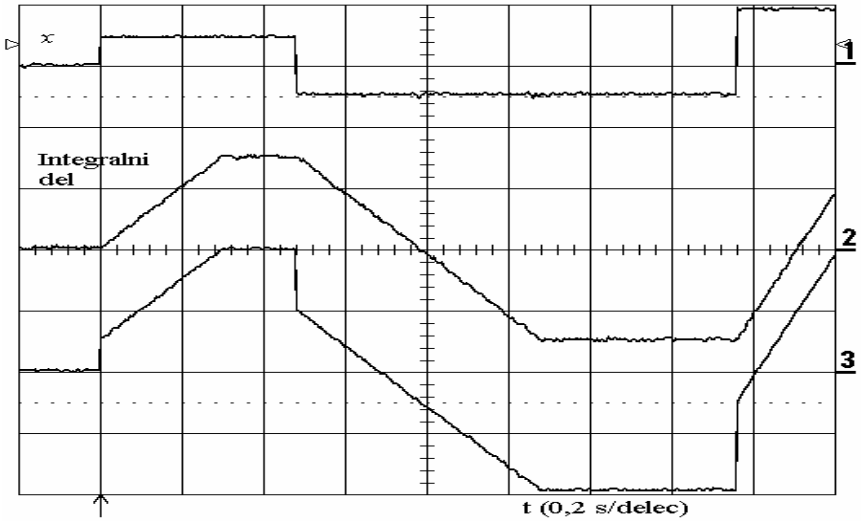
veliki vrednosti ali konstantnem predznaku vhoda) ali zaradi skupnega delovanja, to je njune vsote.

Potreba po omejevanju integralnega člena se pojavlja tudi ob nasičenju izvršnih elementov regulacijskega kroga (t.i. "wind-up"). Le-ti ne morejo vplivati na zmanjšanje regulacijskega pogreška, to je vhodne veličine PI regulatorja, zaradi česa bo integralni člen vztrajno povečeval izhodno vrednost. Poleg preprečevanja presežka je tukaj prisoten tudi problem vztrajnosti integratorja. Če npr. pride do nasičenja izhodnega signala iz PI regulatorja, se mora načeloma ustaviti tudi nadaljnje integriranje (slika 10. 28). V nasprotnem primeru bi I del še naprej integriral in bi pri hitri spremembi predznaka vhodnega signala na spremembo reagiral z zamudo, saj se je medtem "oddaljil" od delovno koristnega območja. Sliki 10. 30 in 10. 31 kažeta blokovni shemi možnih realizacij PI člena. [30].



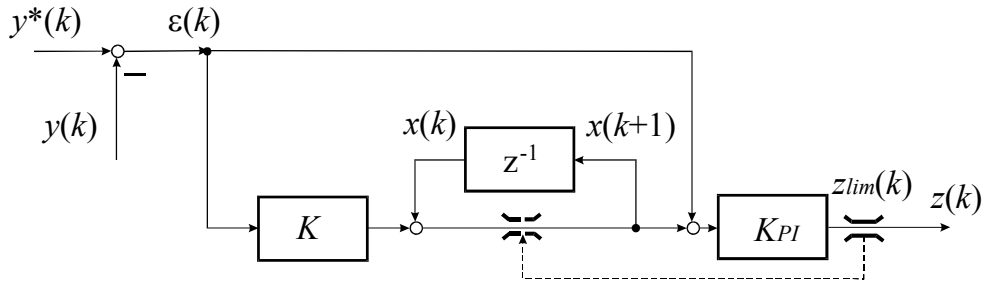
Slika 10. 28: Limitiranje integralnega dela PI člena

Oscilogram odziva regulatorja na različne stopničaste funkcije je na sliki 10. 29.

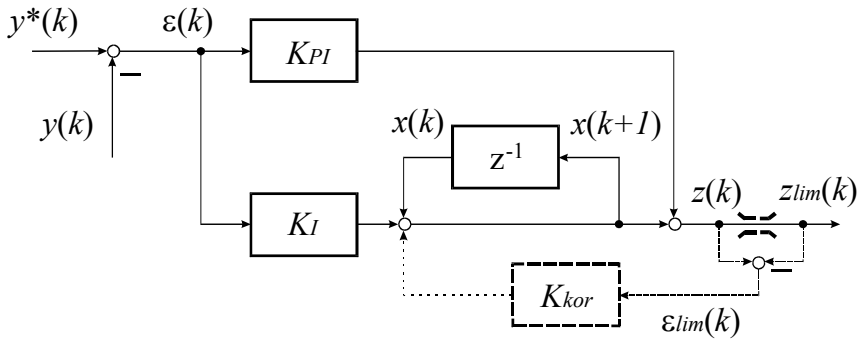


Slika 10. 29: Oscilogram odziva PI regulatorja (K_P = 1, T_i = 100 ms)

V prvem primeru realizacije PI člena na sliki 10. 30 imamo opravka z interakcijo med omejitvijo I člena ter celotnega PI člena ($K = T_{ip}^{-1}$, glej tudi sliko 10. 23). Na primeru s slike 10. 31 pa imamo opravka z dodatnim korekcijskim faktorjem, ki je aktiven ob nasičenju izhodne vrednosti PI člena.



Slika 10. 30: Blokovna shema možne rešitve PI člena (1)



Slika 10. 31: Blokovna shema možne rešitve PI člena (2)

10.3.5.7 Pulzno-širinska modulacija

Ojačenje izhodnega signala iz mikroračunalnika v elektromotorskih pogonih in podobnih vezjih močnostne elektronike je posebno poglavje, s katerim se tukaj ne bomo podrobno ukvarjali. V večini primerov uporabljamo tranzistorske ali tiristorske mostiče, ki delujejo v stikalnem režimu. Primer takega pretvorniškega vezja je prikazan na sliki 10. 5. Izmenično trifazno napetost poljubne oblike generiramo s programiranim spreminjanjem vklopnega časa posameznih tranzistorjev. V ta namen najpogosteje uporabljamo *pulzno-širinsko modulacijo* (angl. Pulse Width Modulation - PWM).

10.3.5.7.1 Prostorski vektor

Napetost trifaznih simetričnih sistemov najlažje opišemo s *prostorskim vektorjem* (angl. space vector), ki je vsota prostorskih prispevkov vseh treh faz:

$$\vec{u}_S = \underbrace{u_{S1} \cdot e^{j0}}_{\vec{u}_{S1}} + \underbrace{u_{S2} \cdot e^{j\frac{2\pi}{3}}}_{\vec{u}_{S2}} + \underbrace{u_{S3} \cdot e^{j\frac{4\pi}{3}}}_{\vec{u}_{S3}}, \qquad (10.\ 15)$$

kjer so u_{S1}, u_{S2} in u_{S3} trenutne vrednosti faznih napetosti.

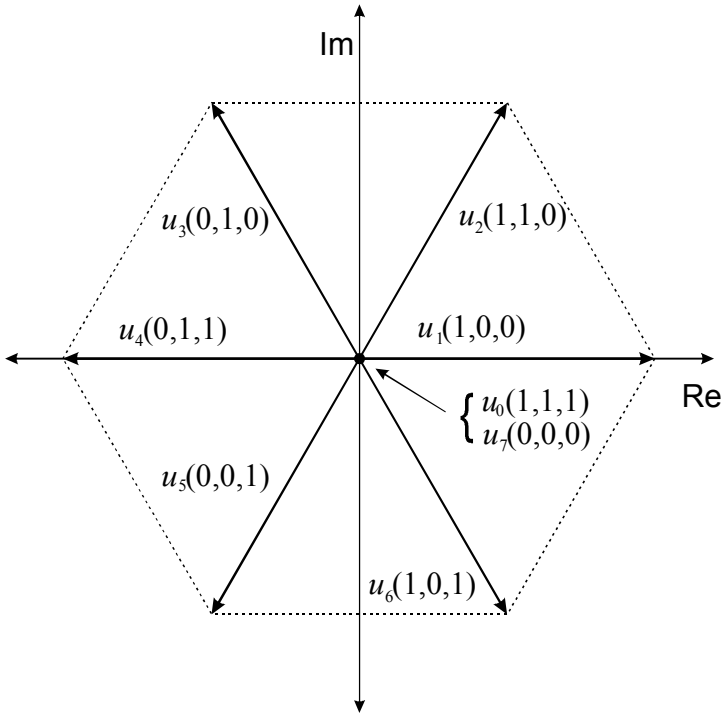
Zaradi diskretnega obratovanja imamo na voljo le omejeno število kombinacij, ki popisujejo stanje šestih tranzistorjev in faznih napetosti. Tranzistorji vklapljajo ali izklapljajo enosmerno napetost vmesnega tokokroga na navitje (u_{vt} na sliki 10. 5). Seveda v nobenem primeru ne smemo hkrati vklopiti obeh tranzistorjev v posamezni veji, saj bi to pomenilo kratek stik med sponkama u_{vt}. Pri vklopljenem zgornjem tranzistorju mora biti spodnji izklopljen, in obratno. Zato ponavadi prikazujamo le stanja zgornjih tranzistorjev, kjer "1" pomeni, da je tranzistor vklopljen, "0" pa, da je izklopljen. Število možnih kombinacij tranzistorskih stanj je $2^3 = 8$: 000, 001, 010, 011, 100, 101, 110 in 111.

Prvi bit se nanaša na stanje tranzistorjev S1 in S4, drugi na S2 in S5, tretji pa na S3 in S6.
Splošni zapis prostorskih vektorjev se glasi (glej tudi sliko 10. 32)

$$\vec{u}_n = u_{vt} \cdot e^{j\frac{(n-1)\pi}{3}}, \qquad \text{pri } n = 1...6$$

ter

$\vec{u}_0 = \vec{u}_7 = \vec{0}$, ker je v tem primeru razlika potencialov med sponkami statorskih navitij enaka nič.

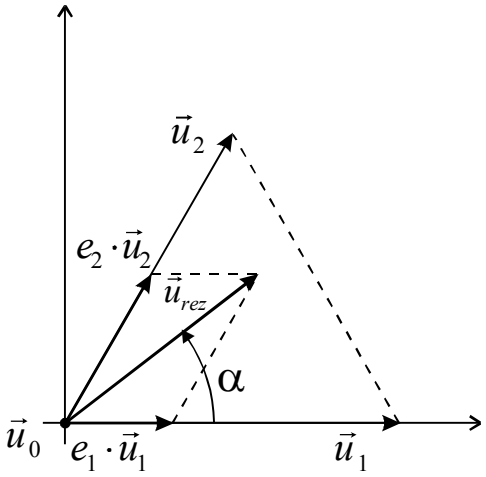


Slika 10. 32: Prostorski vektor v odvisnosti od stanja stikal

Osnovna naloga modulacije je generiranje poljubnega vektorja napetosti znotraj šestkotnika na sliki 10. 32 oz. tri fazne napetosti iz (10. 15) iz 6+2 vektorjev. To dosežemo s preklapljanjem med različnimi kombinacijami, kjer so sekvence prelopov in njihovo trajanje odvisne od izbranega načina modulacije. V tem poglavju bomo prikazali generiranje prostorskega vektorja napetosti prek ene od inačic pulzno-širinske modulacije.

10.3.5.7.2 Naloga procesorja pri PWM

Kot primer si oglejmo stanje na sliki 10. 33. Treba je rešiti problem generiranja vektorja $\vec{u}_{rez}$. Vektor se nahaja v prvi šestini diagrama, torej je logično, da bosta pri njegovem oblikovanju sodelovala sosednja vektorja $\vec{u}_1$ in $\vec{u}_2$ (kombinacije stikal treh faz (1,0,0) in (1,1,0)), kakor tudi $\vec{u}_0 = \vec{u}_7 = \vec{0}$ ((0,0,0) ali (1,1,1)).



Slika 10. 33: Realizacija poljubnega vektorja napetosti

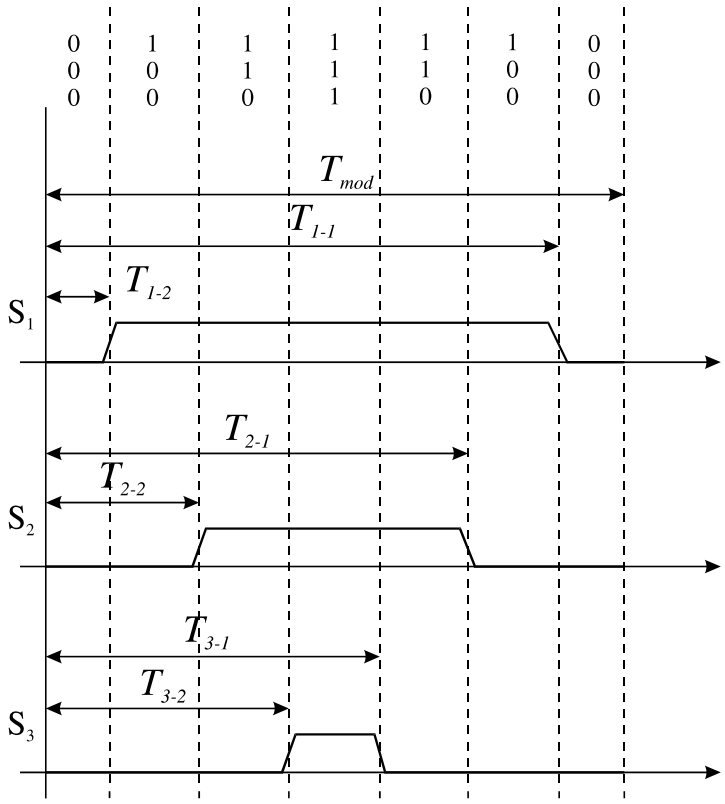
Poleg enosmerne napetosti vmesnega tokokroga u_{vt} je najpomembnejši parameter pri PWM frekvenca oz. interval modulacije T_{mod}. Znotraj tega intervala definiramo tudi časa vklopov T_1 in T_2 za vektorja $\vec{u}_1$ in $\vec{u}_2$. Deleža posameznih vektorjev lahko definiramo s faktorjema e_1 in e_2:

$$e_1 = \frac{T_1}{T_{\text{mod}}}, \qquad e_2 = \frac{T_2}{T_{\text{mod}}}. \tag{10. 16}$$

Oba vektorja bosta znotraj intervala T_{mod} aktivno prispevala k oblikovanju povprečne vrednosti, to je rezultantnega vektorja $\vec{u}_{rez}$:

$$\vec{u}_{rez} = u_{rez} \cdot e^{j\alpha} = e_1 \cdot \vec{u}_1 + e_2 \cdot \vec{u}_2 .$$

Na naslednji sliki je prikazan možen način proženja tranzistorjev za zgoraj opisani primer ter intervala vklopov obeh vektorjev. Takoj vidimo, da pri generiranju rezultante sodeluje tudi ničelni vektor. S spreminjanjem e_1 in e_2 oz. T_1 in T_2 lahko dosežemo poljuben vektor v trikotniku, ki je omejen z dvema kombinacijama stikal.



$$T_1 = (T_{2\text{-}2} - T_{1\text{-}2}) \text{ x } 2$$

$$T_2 = (T_{3\text{-}2} - T_{2\text{-}2}) \text{ x } 2$$

Slika 10. 34: Sekvenca vklopov tranzistorjev v prvi šestini šestkotnika

Sekvenca vklopov je naslednja: $\vec{u}_0 \rightarrow \vec{u}_1 \rightarrow \vec{u}_2 \rightarrow \vec{u}_7 \rightarrow \vec{u}_2 \rightarrow \vec{u}_1 \rightarrow \vec{u}_0$ oz. kombinacije stikal $(0,0,0) \rightarrow (1,0,0) \rightarrow (1,1,0) \rightarrow (1,1,1) \rightarrow (1,1,0) \rightarrow (1,0,0) \rightarrow (0,0,0)$.

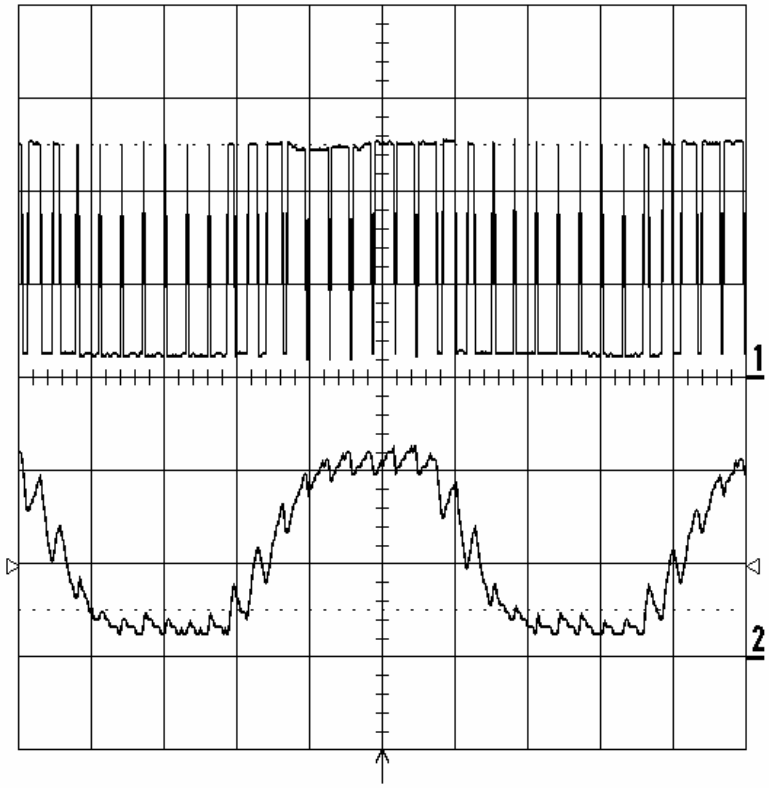
Mikroračunalnik mora pri generiranju signala za proženje tranzistorjev izvajati naslednje naloge:

1. Iz želenih napetosti izračunati velikost in smer želenega prostorskega vektorja. To pa hkrati pomeni detekcijo šestine, v kateri se ta vektor nahaja oz. kombinacije stikal dveh sosednjih vektorjev.
2. Izračunati trajanja vklopov posameznih vektorjev (en. (10. 16)).
3. Generirati vklopne sekvence s slike 10. 34.

Poleg tega je pri sodobnih mikrokontrolerjih (npr. TMS320F240) možna tudi *kompenzacija mrtvega časa.* Ob izračunih časov preklopov posameznih tranzistorjev namreč običajno predpostavljamo takojšnji vklop ali izklop, v praksi pa do tega pride z določeno zakasnitvijo. Posebej pri nizkih napetostih lahko ti mrtvi časi vplivajo na to, da je dejanska napetost občutno nižja od želene in predpostavljene.

Slika 10. 35 kaže oscilograma dejanske in filtrirane vrednosti krmilnega signala za enega od tranzistorjev mostiča ob uporabi PWM. V primeru je zaradi boljše vizualne ločljivosti pulzov

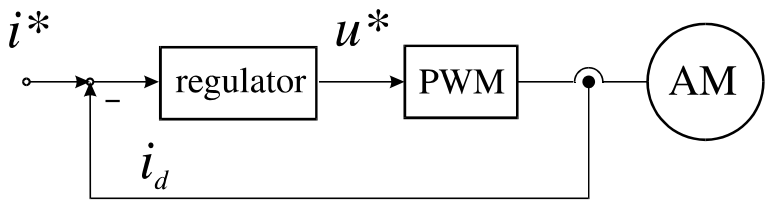
namenoma izbrana previsoka frekvenca želene sinusne napetosti (350 Hz), ki pri pogonih le redko pride v poštev. Zaradi tega je tudi filtrirana vrednost, ki naj bi ustrezala sinusu, nekoliko nazobčana. Frekvenca modulacije je 6,7 kHz (perioda 150 μs).



Slika 10. 35: Oscilogram prožilnih pulzov enega tranzistorja kot posledica PWM ter njihova filtrirana vrednost (čas. baza 0,5 ms)

10.3.5.7.3 Tokovni vir

Nekatere aplikacije (različni tokovni viri, izmenični motorji itd.) zahtevajo tokovno napajanje. Zato je pri sistemih s PWM potrebno zaključiti tokovno zanko, ki je v večini primerov skrajnja notranja zanka takšnih kaskadnih regulacij, s pa tem tudi najhitrejša (npr. T_V = 100μs). Izhod iz regulatorja (običajno PI) toka je želena napetost, ki jo PWM blok mora realizirati.



Slika 10. 36: Tokovna zanka pri PWM