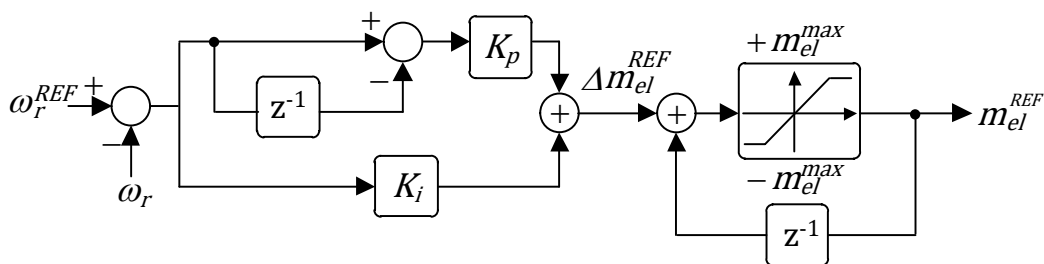
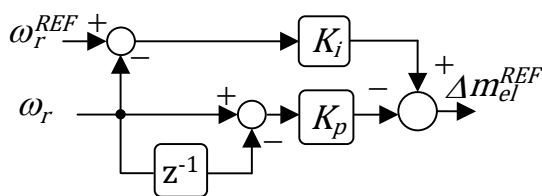


Blok dijagram diskretnog regulatora u inkrementalnoj formi dat je na slici 4.37.



Slika 4.37. Digitalni PI regulator u inkrementalnoj formi

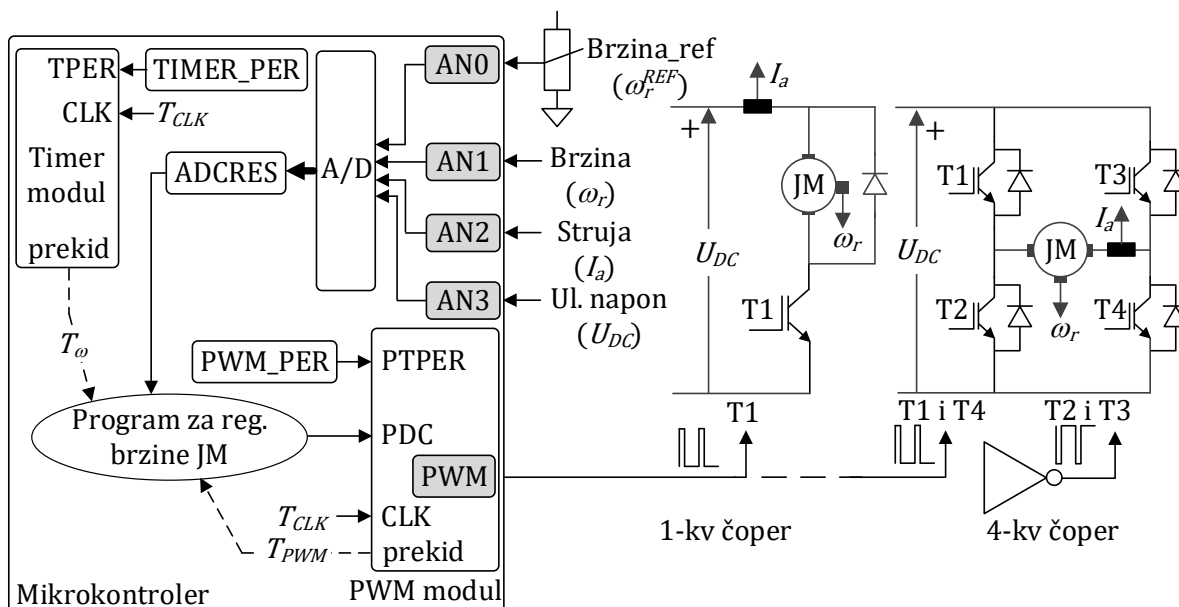
Prikazani regulator je u inkrementalnoj formi sa proporcionalnim dejstvom u direktnoj grani. Moguće poboljšanje je premeštanje proporcionalnog dejstva u lokalnu povratnu spregu, slika 4.38. Ovim se eliminišu nepoželjne nagle promene upravljačke promenljive (zadatog momenta) koje nastaju delovanjem proporcionalnog dejstva na nagle skokovite promene referentne vrednosti (u ovom slučaju brzine rotora).



Slika 4.38. P dejstvo u povratnoj grani

4.9. Mikroprocesorski program za regulaciju brzine JM

Zadatak mikroprocesorskog programa za regulaciju brzine JM je da upravlja DC/DC pretvaračem tako da JM razvije moment potreban da se ostvari i održi željena brzina rotora. Obrađena su dva primera mikroprocesorskog programa za regulaciju brzine, program koji upravlja naponom armature JM, i program koji upravlja strujom armature JM. Oba primera programa podrazumevaju da mikrokontroler poseduje potrebne periferne module (A/D, Timer i PWM) koji su sa pretvaračem i mernim kolima povezani kao što je dato na slici 4.39.



Slika 4.39. Podrazumevane periferije i način povezivanja pinova mikrokontrolera sa jednokvadrantnim (1-kvč) i četvorokvadrantnim (4-kvč) DC/DC čoperom

Opisani programi za regulaciju brzine podržavaju pogon sa JM koji poseduje jednokvadrantni (1-kvč) ili četvorokvadrantni (4-kvč) DC/DC pretvarač. U oba slučaja generiše se samo jedan izlazni PWM signal koji se u slučaju 4-kv čopera invertuje i komplementira.

Mikrokontroler poseduje A/D konvertor sa bar četiri analogna ulaza koji radi unipolarno sa mernim opsegom od 0 do 5V. Referentna vrednost brzine rotora ω_r^{REF} se zadaje kao analogni naponski signal na analognom ulazu A/D konvertora AN0, na primer promenom položaja potencijometra. Signal proporcionalan brzini rotora ω se nakon potrebnih prilagođenja dovodi na analogni ulaz AN1 (0 do ω_r^{MAX} se pretvara u 0 do 5 V), slično, signal proporcionalan struji armature I_a se dovodi na analogni ulaz AN2 (za 1-kv čoper struja od 0 do I_a^{MAX} a za 4-kv čoper struja $\pm I_a^{MAX}$ se pretvara u signal od 0 do 5 V), a signal proporcionalan ulaznom DC naponu U_{DC} pretvarača se dovodi na analogni ulaz AN3 (0 do U_{DC}^{MAX} se pretvara u signal od 0 do 5 V).

Mikrokontroler poseduje brojački modul koji radi u modu brojanja vremena (Timer), sa periodom internog sata (clock signala) T_{CLK} i kome se vremenski period brojanja zadaje putem TPER registra. Timer modul može da generiše prekid po isteku vremena $T_{CLK} \cdot TPER$.

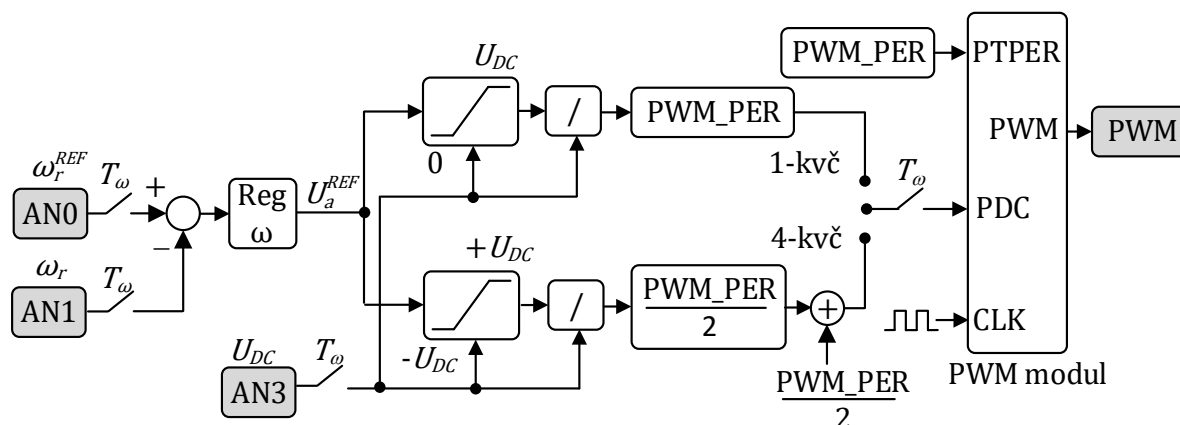
Mikrokontroler poseduje PWM modul koji radi sa periodom clock signala T_{CLK} , perioda PWM signala se zadaje putem PTPER registra a faktor ispune putem PDC registra. PWM modul može da generiše prekid po isteku vremena $T_{CLK} \cdot PTPER$.

U nastavku će biti opisane dve verzije programa za regulaciju brzine JM. Prva verzija programa ne zahteva strujnu kontrolu, čini je samo spor regulator brzine koji sa relativnom dugom periodom T_ω upravlja naponom armature, i prosta je za mikroprocesorsku realizaciju. Druga verzija uključuje i strujnu kontrolu, tj. brzu promenu napona u cilju dostizanja referentne struje, i sastoji se od složenije kaskadne strukture koju čine spor regulator brzine (perioda T_ω) i mnogo brži regulator struje (perioda T_{PWM}). Iako složenija, regulacija brzine upravljanjem strujom se češće koristi jer čini JM idealnim konvertorom momenta, što je osnovni uslov za optimalnu regulaciju brzine ili položaja rotora.

4.9.1. Mikroprocesorski program za regulaciju brzine JM upravljanjem naponom

U ovom poglavlju će biti opisan relativno prost mikroprocesorski program za digitalnu regulaciju brzine JM bez regulacije struje, samo sa upravljanjem srednjom vrednošću impulsnog napona na izlazu prekidačkog DC/DC pretvarača. Pošto se ne koristi brza kontrola struje program se može izvršavati u realnom vremenu i na mikrokontroleru niskih performansi.

Uprošćen blok dijagram programa je dat na slici 4.40.



Slika 4.40. Blok dijagram regulacije brzine JM upravljanjem naponom [4_4]

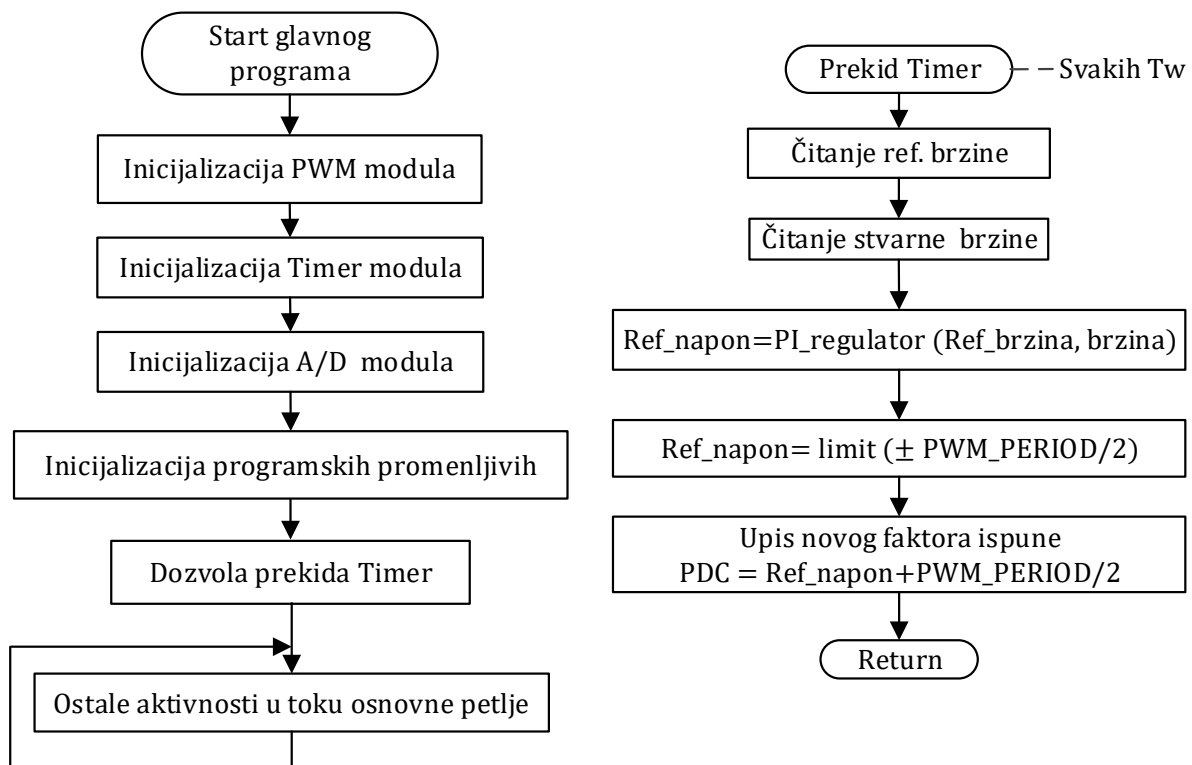
Mikrokontroler sa relativno dugačkom periodom T_ω (reda veličine 1ms -10ms) odabira zadatu i merenu brzinu i računa signal greške. Na osnovu te greške regulator brzine izračunava potreban napon armature U_a^{REF} koji se u slučaju 1-kvč limitira na opseg 0 do U_{DC} , dok u slučaju 4-kvč na opseg $\pm U_{DC}$. Na osnovu referentnog napona U_a^{REF} i merenog napona U_{DC} se dalje u zavisnosti od tipa pretvarača računa vrednost za upis u PDC registar. Promena u PDC registru postaje aktuelna u toku sledeće PWM periode, u kojoj dolazi do promene faktora ispune PWM signala, tj. do promene srednje vrednosti napona armature JM koji reaguje promenom brzine.

4.9.1.1. Uprošćen algoritam programa za regulaciju JM upravljanjem naponom

Na slici 4.41. je dat uprošćen blok dijagram algoritma programa za regulaciju brzine JM upravljanjem naponom koji sadrži samo neophodne operacije i raspored njihovog pozivanja. Algoritam se sastoji iz glavnog dela (inicijalizacija i glavna beskonačna programska petlja), i prekidne rutine brojačkog Timer modula.

Prilikom inicijalizacije se podešavaju PWM modul (da generiše PWM signal željenih parametara), A/D modul (da merne signale pretvori u digitalne ekvivalente) i Timer modul (da meri i signalizira istek periode brzinske petlje T_ω). Takođe, tokom inicijalizacije se razne programske varijable postavljaju na potrebno početno stanje. Glavni program nakon toga ulazi u beskonačnu petlju u kojoj čeka prekid brojačkog modula. Ali, to obično nije tako, vreme između dva prekida se obično iskoristi za izvršenje neke od asinhronih funkcija.

U prekidnoj rutini Timer program jednom u periodu T_ω čita referentnu i merenu brzinu, računa i limitira referentni napon i upisuje odgovarajući faktor ispune u PDC registar.



Slika 4.41. Uprošćen blok dijagram programa za regulaciju brzine JM upravljanjem naponom armature (primer samo za 4-kvč)

4.9.1.2. Detaljniji blok dijagram algoritma mikroprocesorskog programa za regulaciju brzine JM upravljanjem naponom armature

U ovom poglavlju je dat detaljniji blok dijagram algoritma mikroprocesorskog programa za regulaciju brzine JM upravljanjem naponom armature. Program podrazumeva iste referentne i merne signale, i jedan PWM izlaz koji su sa pretvaračem povezani kao na slici 4.39.

Blok dijagram algoritma glavnog programa izvršava dva osnovna zadatka i dat je na slici 4.42. Prvi zadatak, inicijalizaciju potrebnih perifernih modula mikrokontrolera glavni program izvršava samo jednom, odmah nakon resetu. Inicijalizacija brojačkog modula koji treba da generiše prekid po isteku periode brzinske petlje T_ω obuhvata podešavanje periode rada brojača (npr. za $T_\omega = 1$ ms, pri $f_{clk} = 30$ MHz u registar TPER se upisuje konstanta $TIMER_PER = 30 \text{ MHz}/1 \text{ kHz} = 30000$), način rada (*autoreload*) i dozvolu rada brojačkog modula. Inicijalizacija PWM modula obuhvata podešavanje perioda PWM signala (npr. za $T_{PWM} = 50$ μ s, pri $f_{clk} = 30$ MHz u registar PTPER se upisuje konstanta $PWM_PER = 30 \text{ MHz}/20 \text{ kHz} = 1500$), postavljanje inicijalne vrednosti faktora ispune koja odgovara početnom naponu jednakom nuli, $PDC = 0$ (za 1-kv čoper) ili $PDC = PWM_PER/2$ (za 4-kv čoper), izbor testere kao tipa PWM nosioca, i dozvolu rada PWM modula. Inicijalizacija A/D modula obuhvata definisanje ulaza koji će se koristiti (AN0, AN1 i AN3), načina startovanja konverzije, definisanje brojnog formata rezultata i dozvolu rada A/D modula. Inicijalizacija hardvera se uobičajeno završava podešavanjem multipleksiranih pinova kontrolera, u konkretnom slučaju tri pina su povezana sa A/D konvertorom i označena kao analogni ulazi, a jedan povezan sa PWM modulom i označen kao digitalni izlaz. U okviru prvog zadatka inicijalizuju se i programski moduli i definišu razne globalne varijable. Na primer, integralnu sumu PI regulatora brzine treba postaviti na početnu vrednost nula kako bi se po pokretanju programa izbegao skok napona na motoru, a slično, kod PI regulatora u inkrementalnoj formi, treba resetovati prethodnu grešku brzine. Nakon inicijalizacije, program dozvoljava prekid brojačkog modula (program ne koristi PWM prekid, te on ostaje zabranjen) i ulazi u beskonačnu petlju.

Blok dijagram algoritma programa prekidne rutine brojačkog modula je prikazan na slici 4.43. Ovaj deo programa je centar regulacije brzine JM upravljanjem naponom i čini ga skup funkcija koje treba da se izvršavaju sa periodom T_ω u prekidnoj rutini brojačkog modula:

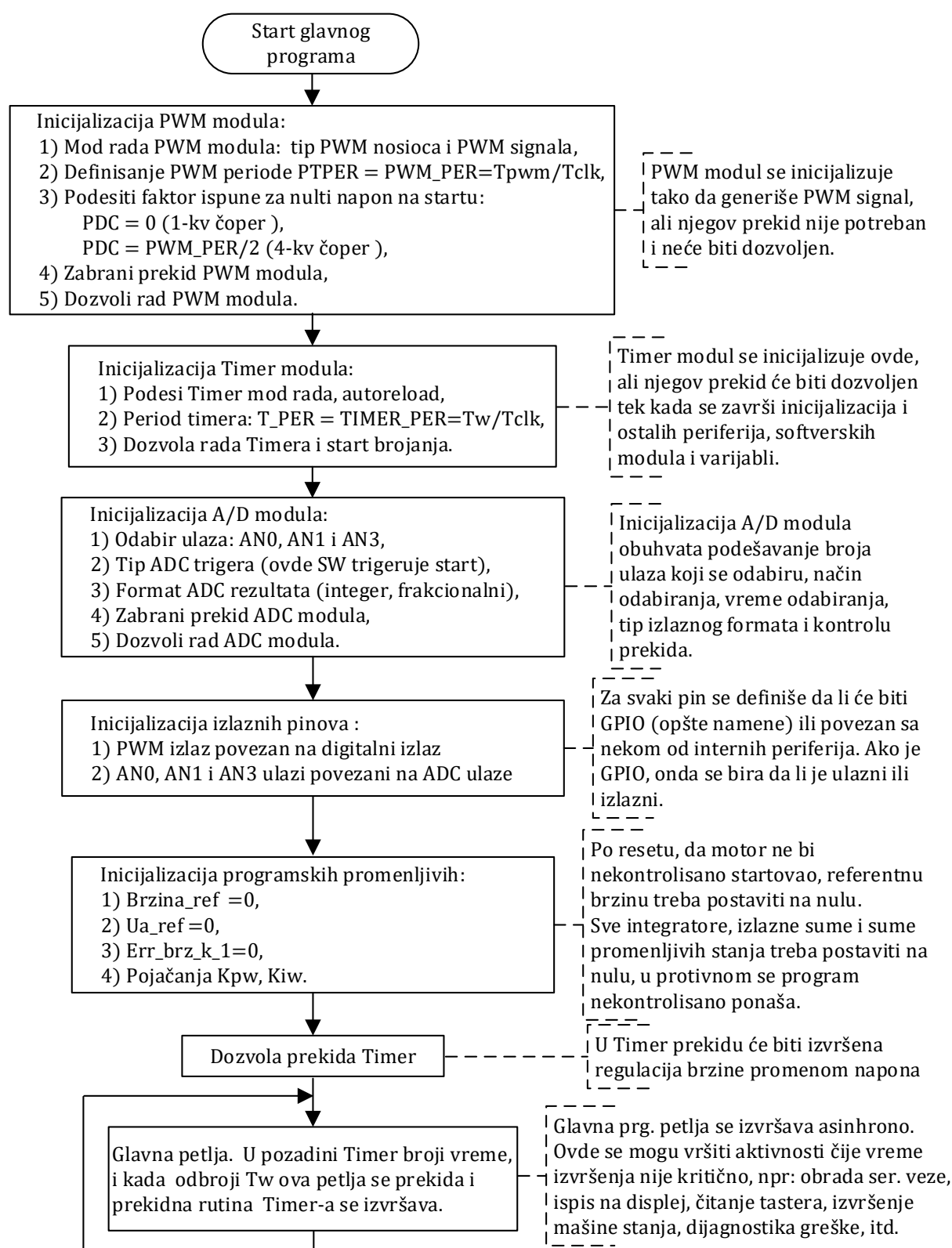
- Uzimanje najnovijeg odbirka brzine rotora, pogl. 4.7.1,
- Uzimanje najnovijeg odbirka ulaznog napona, pogl. 4.7.3,
- Čitanje referentne vrednosti za brzinu rotora,
- Rampa referentne brzine (prikazana ali nije korišćena u daljem algoritmu),
- Račun diskretne jednačine PI regulatora brzine, poglavlje 4.8,
- Referentni napon na izlazu PI regulatora brzine se limituje:
 - u slučaju jednokvadrantnog čopera na opseg $0 - U_{DC}$,
 - u slučaju četvorokvadrantnog čopera na opseg $\pm U_{DC}/2$.
- Ovako limitovan napon se koristi za račun potrebnog faktora ispune:
 - u slučaju jednokvadrantnog čopera, u trenutku kT_ω važi:

$$PDC(k) = PWM_PER \cdot u_a^{REF}(k)/U_{DC}(k), \quad (4.68)$$

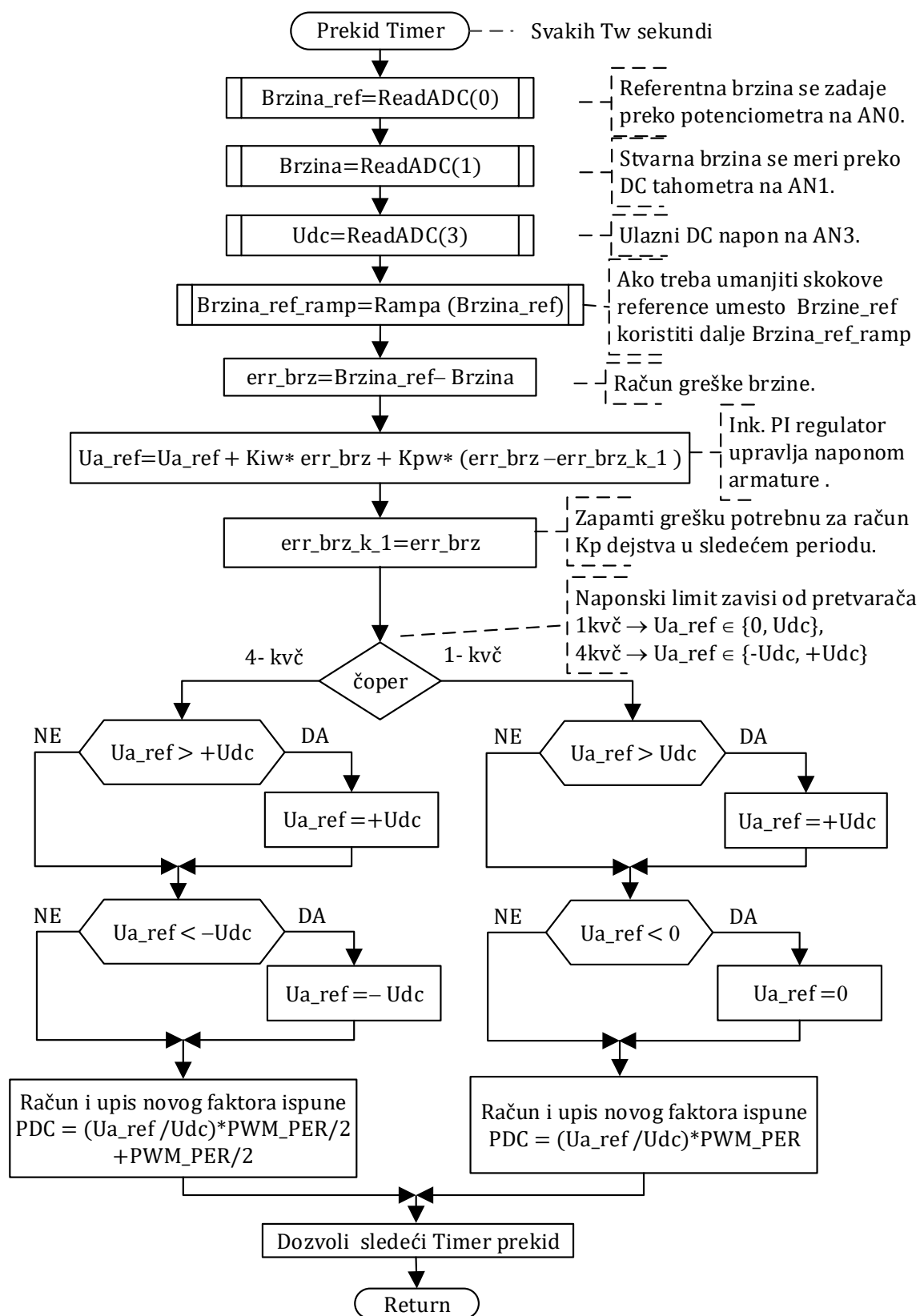
- u slučaju četvorokvadrantnog čopera,

$$PDC(k) = (PWM_PER/2) \cdot u_a^{REF}(k)/U_{DC}(k) + PWM_PER/2. \quad (4.69)$$

Nakon završetka prekidne rutine program se vraća u beskonačnu petlju glavnog programa i čeka novi prekid brojača.



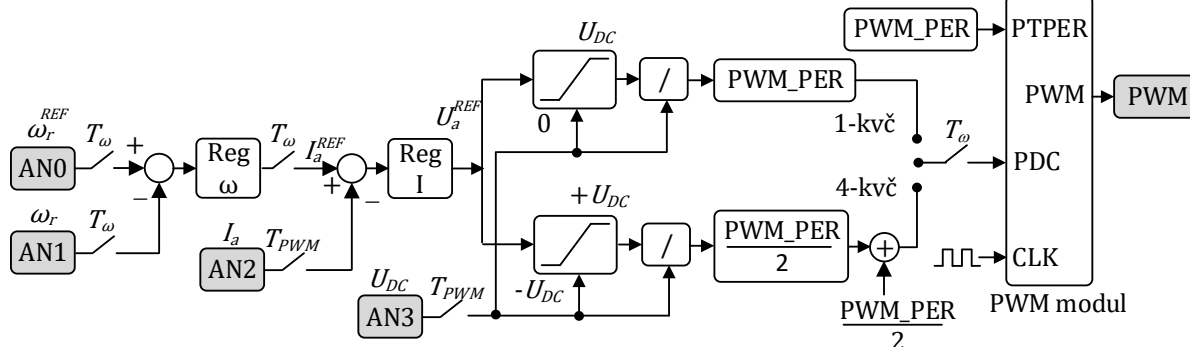
Slika 4.42. Blok dijagram algoritam glavnog programa za regulaciju brzine JM upravljanjem naponom armature



Slika 4.43. Blok dijagram algoritma programa prekidne rutine brojačkog modula za regulaciju brzine JM upravljanjem naponom armature (brzinska petlja)

4.9.2. Mikroprocesorski program za regulaciju brzine JM upravljanjem strujom

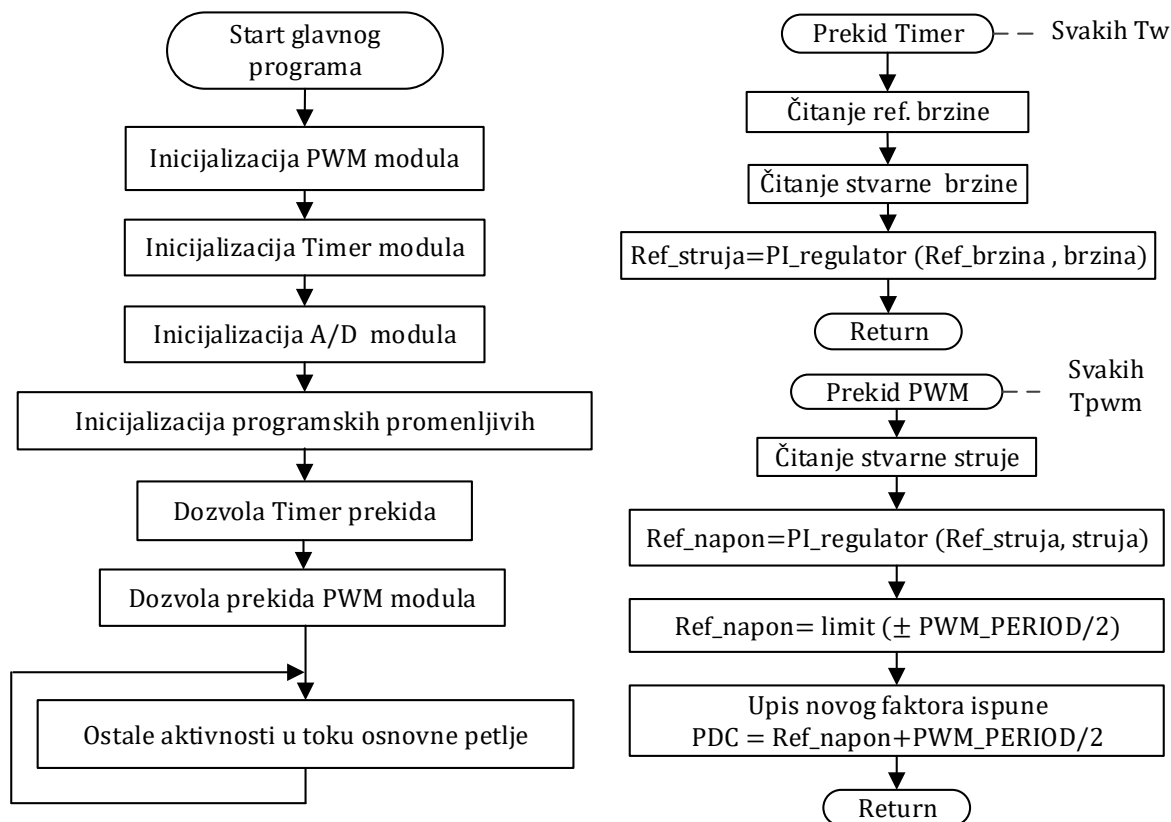
U ovom poglavlju je opisan složeniji, ali sa stanovišta regulacije efikasniji program za regulaciju brzine JM. Po ovom rešenju regulator brzine upravlja strujom armature i time praktično direktno kontroliše moment JM. Regulacija je kaskadnog tipa, regulator brzine periodom T_ω menja referentnu struju armature, a podređeni strujni regulator najbrže moguće, svake T_{PWM} , upravlja naponom armature tako da se ta struja što pre i ostvari. Programski i periferni blokovi koji se koriste za ovakav tip regulacije brzine prikazani su na slici 4.44.



Slika 4.44. Blok dijagram regulacije brzine JM upravljanjem strujom armature [4_4]

4.9.2.1. Uprošćen algoritam programa za regulaciju brzine JM upravljanjem strujom

Na slici 4.45 je dat uprošćen blok dijagram algoritma za kaskadnu regulaciju brzine JM upravljanjem strujom. Program ima glavni deo bez bitnih operacija, prekidnu Timer rutinu koja generiše referentnu struju i prekidnu PWM rutinu koja menja napon da se ta struja i ostvari.



Slika 4.45. Uprošćen blok dijagram programa za regulaciju brzine JM upravljanjem strujom

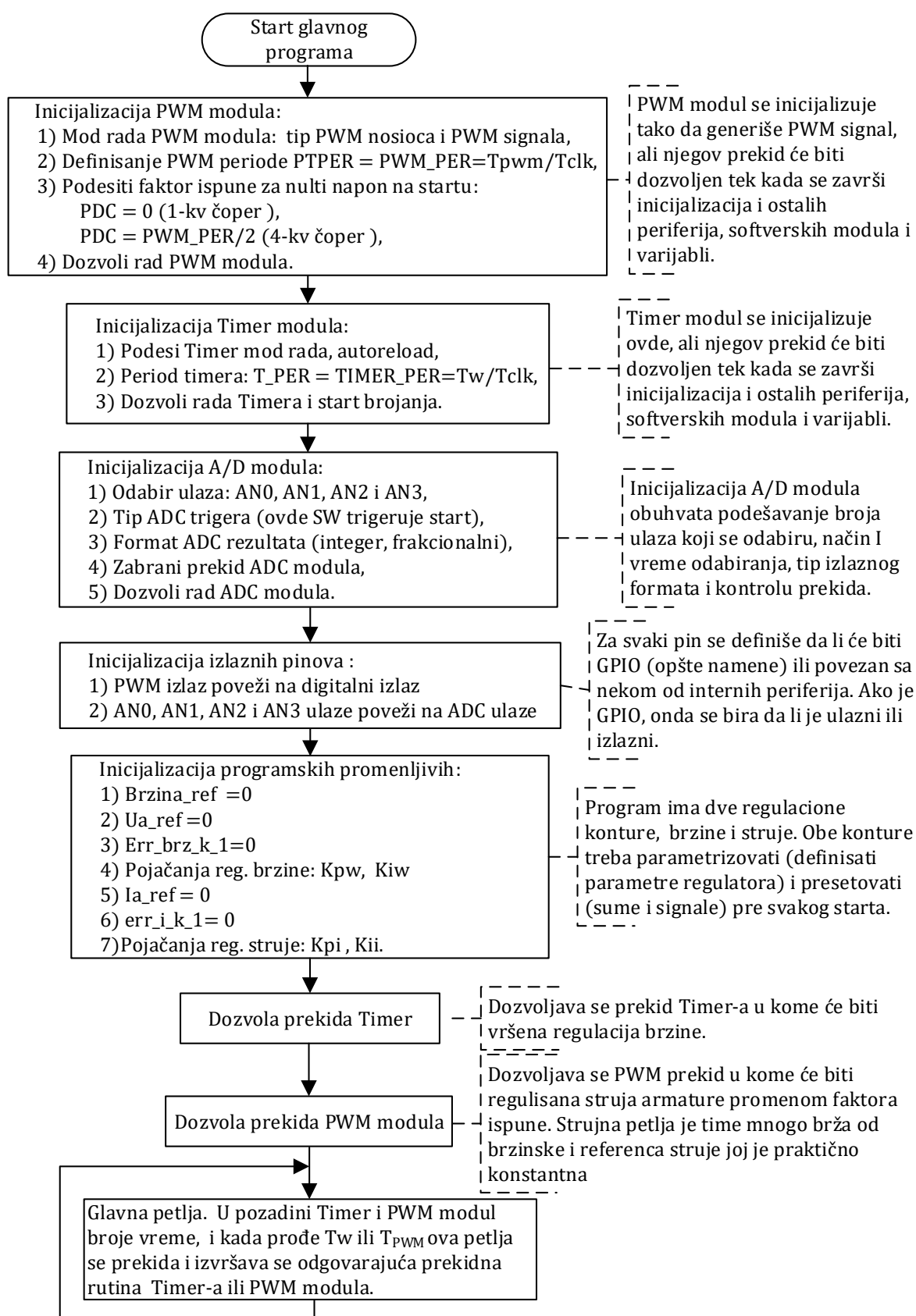
4.9.2.2. Detaljniji blok dijagram algoritma mikroprocesorskog programa za regulaciju brzine JM upravljanjem strujom armature

U ovom poglavlju je dat mikroprocesorski program za regulaciju brzine JM upravljanjem strujom armature koji podrazumeva isti mikrokontroler, kao i iste ulazno/izlazne signale na njegovim pinovima, kao što je već prikazano na slici 4.39. Program ima tri jasno razdvojena zadatka i prikazan je u tri dela: 1. glavni deo (slika 4.46) u kome se nakon reseta podešavaju svi potrebni periferni i programski moduli, te ulazi u beskonačnu petlju, 2. prekidna rutina brojačkog modula (slika 4.47) u kojoj se u skladu sa odabranom periodom odabiranja T_{ω} izvršava zakon upravljanja brzinom, i 3. prekidna rutina PWM modula (slika 4.48) u kojoj se u skladu sa odabranom PWM periodom T_{PWM} izvršava zakon upravljanja strujom armature.

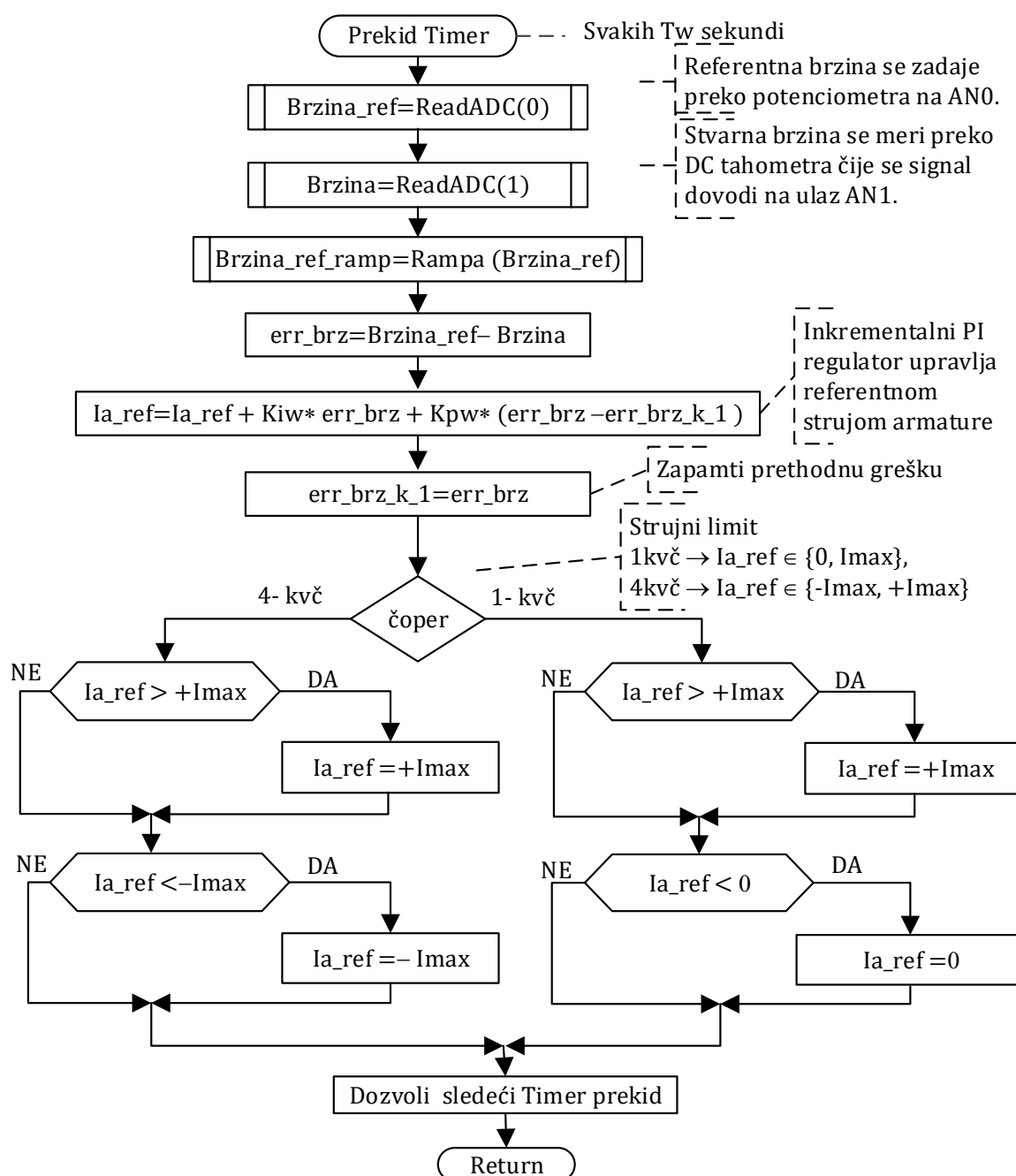
Blok dijagram glavnog programa je dat na slici 4.46. Inicijalizacija u glavnom programu je slična kao u slučaju regulacije brzine JM upravljanjem naponom, slika 4.42, s tim što je sada neophodno po završetku inicijalizacije pored prekida brojačkog modula dozvoliti i prekid PWM modula radi izvršenja brže strujne regulacione petlje. Takođe, sada je potrebno parametrizovati i PI regulator struje i presetovati ga na potrebne početne vrednosti. Nakon inicijalizacije glavni deo programa i u ovom slučaju ulazi u beskonačnu petlju, u kojoj čeka da se desi jedan od dva prekida, brojača brzinske petlje ili PWM modula strujne petlje. Dodatno, u glavnom programu se mogu raditi i vremenski nekritične operacije kao što je obrada serijske veze.

Blok dijagram algoritma programa prekidne rutine brojačkog modula je dat na slici 4.47. U ovoj prekidnoj rutini se izvršava zakon upravljanja brzinske petlje sa periodom T_{ω} . Prvo se vrši odabiranje referentne i merene brzine sa analognih ulaza AN0 i AN1. Na ovom mestu je moguće realizovati rampu referentne brzine čime se izbegavaju njeni nagli skokovi. U tom slučaju bi se greška brzine dobila kao razlika merene brzine u odnosu na Brzina_ref_ramp, a ne na vrednost Brzina_ref. U nastavku algoritma, na osnovu diskretnih jednačina PI regulatora se sada računa referentna vrednost struje armature, a ne referentni napon, kao što je to bio slučaj sa programom na slici 4.43. Po potrebi, ovaj deo programa limitira zadatu vrednost struje armature u zavisnosti od strujnih mogućnosti pogona. Limit struje I_{max} se definiše na osnovu kritičnije vrednosti struje, bilo maksimalne struje motora ili maksimalne struje korišćenog DC/DC pretvarača. U slučaju jednokvadratnog čopera dozvoljena je samo pozitivna, dok su u slučaju četvorokvadratnog čopera dozvoljene i pozitivna i negativna referentna struja armature. U ovom prekidu izračunata referentna vrednost struje armature ostaje nepromenjena tokom sledećih T_{ω} sekundi, tj. sledećih T_{ω}/T_{PWM} PWM perioda. Ključ efikasnosti kaskadne regulacije je u razdvajanju dinamika spoljne brzinske i podređene unutrašnje strujne petlje. Da bi se ove dinamike razdvojile, potrebno je podesiti da važi $T_{\omega} > 10 \cdot T_{PWM}$, čime se daje vreme od bar deset PWM perioda strujnoj petlji da upravljanjem naponom svake PWM periode tu referentnu struju armature i ostvari, pre nego što se ona po isteku tekuće periode T_{ω} ponovo ne promeni.

Blok dijagram algoritma programa PWM prekidne rutine je dat na slici 4.48. U ovoj prekidnoj rutini se izvršava zakon upravljanja strujom sa periodom T_{PWM} . Program ulazi u prekidnu rutinu na početku svake PWM periode, iniciran prekidom koji generiše interni brojač u sklopu PWM modula. U PWM prekidnoj rutini se prvo odabire merena struja armature (u primeru na slici 4.41 to je analogni ulaz AN2), poredi sa referentnom strujom čija je vrednost postavljena u poslednjem prekidu Timer modula, i računa potreban napon na osnovu diskretnih jednačina PI regulatora. Dobijeni referentni napon armature se u skladu sa ulaznim naponom pretvarača (u primeru se odabira na analognom ulazu AN3) pretvara u odgovarajući faktor ispunje skaliran za upis u PDC registar. Ovim je PWM modulu zadat novi faktor ispunje PWM signala, koji će biti aktivan u sledećem PWM periodu, i dovodi do promene struje armature.

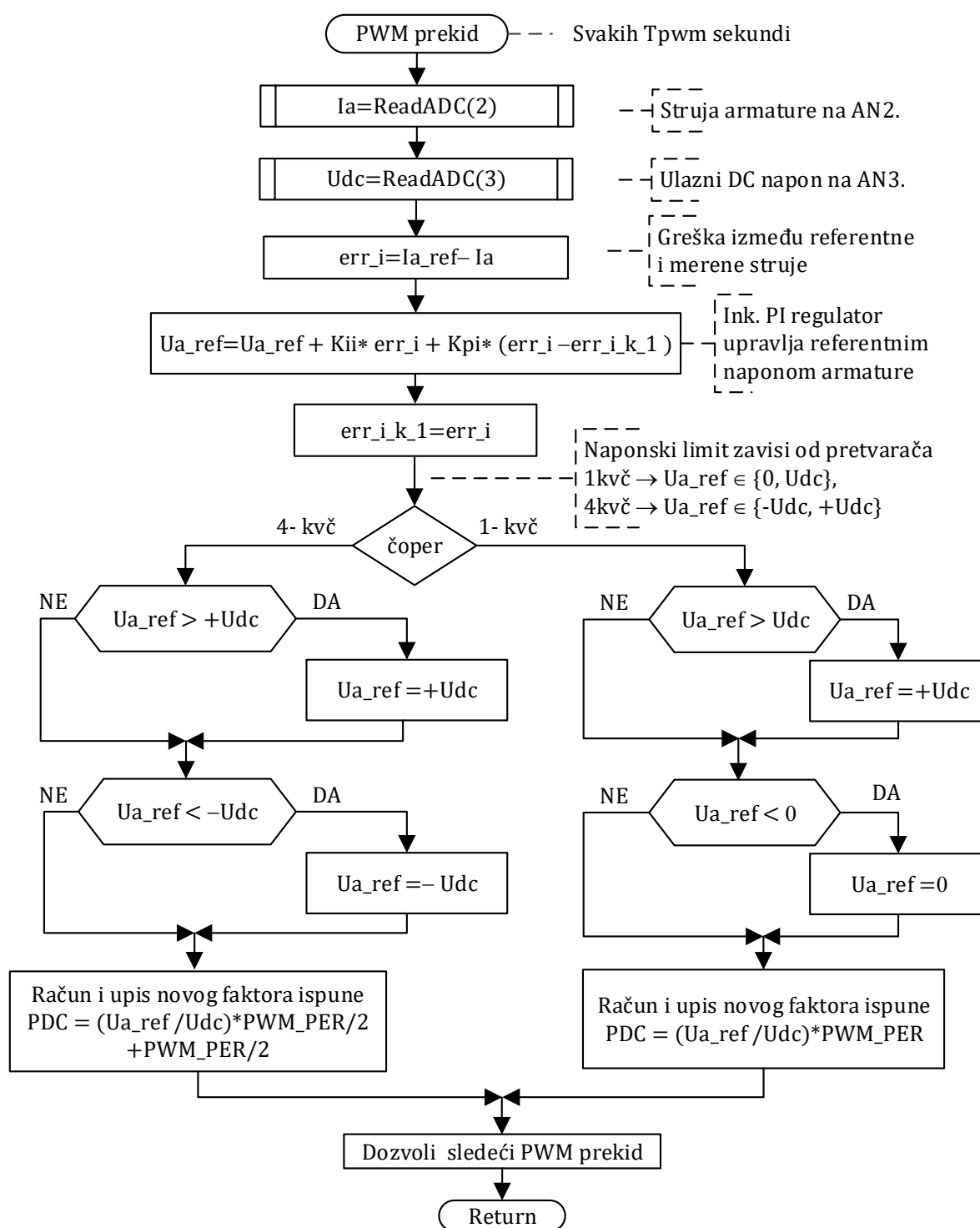


Slika 4.46. Blok dijagram algoritma glavnog programa za regulaciju brzine JM upravljanjem strujom armature



Slika 4.47. Blok dijagram algoritma programa prekidne rutine brojačkog modula za regulaciju brzine JM upravljanjem strujom armature (brzinska petlja)

Program za kaskadnu regulaciju brzine JM upravljanjem strujom armature se može realizovati i samo sa jednom prekidnom rutinom, iniciranom od strane PWM modula, kao je prikazano na slici 4.49. Ovo je čest slučaj u praksi jer omogućuje bolji pregled rada i lakšu sinhronizaciju podređene i nadređene regulacione petlje. U ovom slučaju se prekid brojačkog modula ne koristi, već se duži period odabiranja regulacione konture brzine T_ω definiše kao celi broj proteklih PWM perioda. Na primer, ako je PWM frekvencija 20 kHz (tj. PWM perioda traje $T_{PWM}=50\text{ }\mu\text{s}$) a željeni period brzinske petlje $T_\omega=1\text{ms}$, tada je period T_ω moguće definisati kao vremenski interval koji se sastoji od dvadeset PWM perioda, $T_\omega/T_{PWM} = 20$.



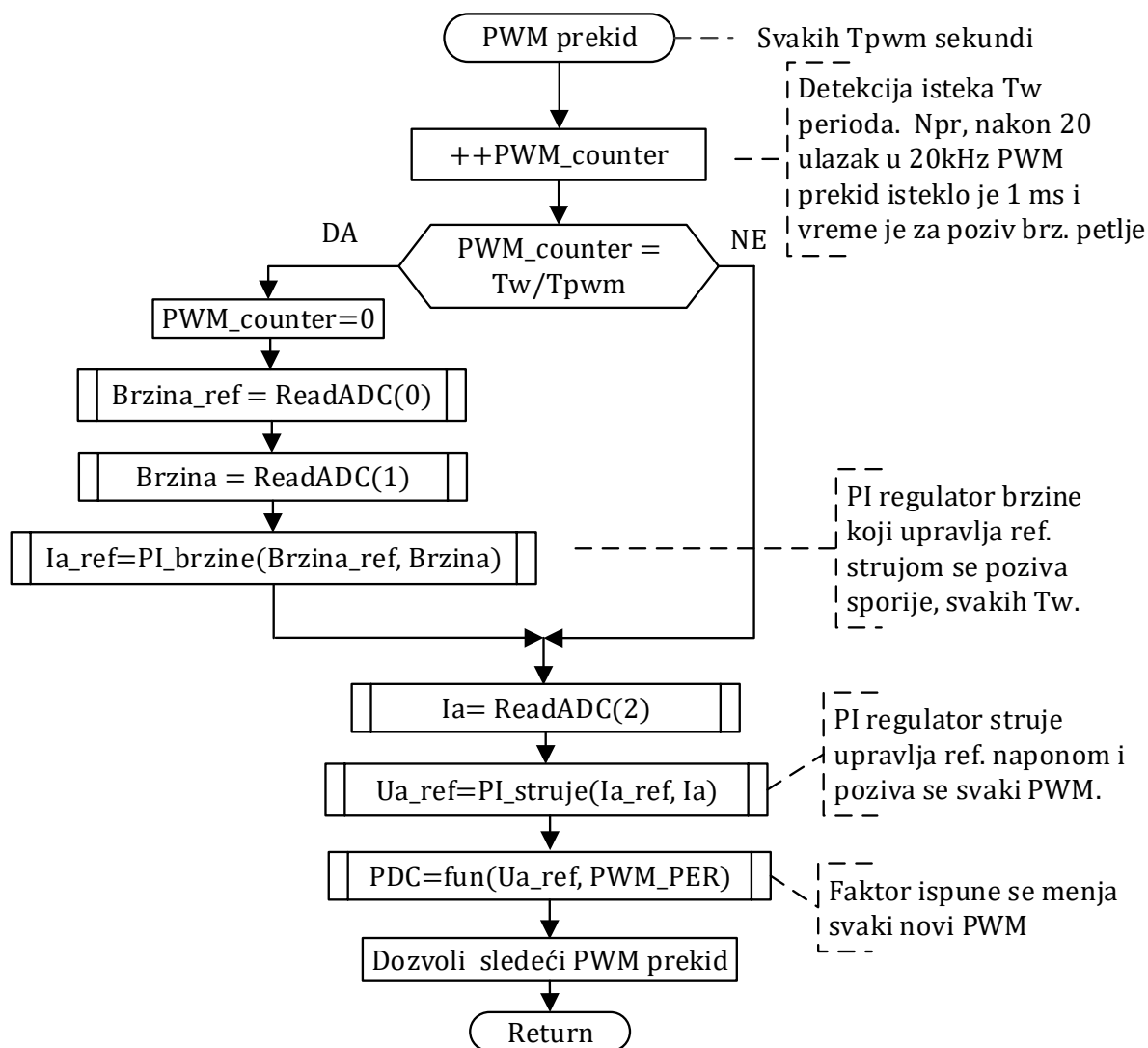
Slika 4.48. Blok dijagram algoritma programa PWM prekidne rutine za regulaciju brzine JM upravljanjem strujnom armature (strujna petlja)

Rešenje programa sa jednim prekidom treba u svakom PWM prekidu da izvršava unutrašnju strujnu regulacionu konturu, tj. diskretne jednačine strujnog regulatora brzine, a u svakom 20-tom PWM prekidu da dodatno računaju i diskretne jednačine regulatora brzine. Isto kao i ranije, regulator brzine na osnovu referentne i merene brzine menja referentnu struju armature koja zatim ostaje nepomenjena do njegovog sledećeg poziva, dakle tokom sledećih

dvadeset PWM perioda. Ovim je ostavljeno dovoljno vremena strujnom regulatoru da tu referentnu struju i ostvari. Regulator struje se poziva u sledećih dvadeset PWM perioda, čita najnoviju vrednost struje, poredi je sa praktično konstantnom poslednje postavljenom referentnom vrednošću struje, i promenom napona pokušava da je što pre i ostvari. Nakon prolaska dvadeset PWM perioda, regulator brzine se ponovo poziva, računa novu referentnu vrednost struje, koju regulator struje već u nastavku periode preuzima i počinje da prati.

Prednost ove strukture je što regulacione konture više ne prekidaju jedna drugu. Takođe, pozivi regulatora brzine i podređenog strujnog su savršeno sinhronizovani, tj. odmah po pozivu regulatora brzine i postavljanja nove referente vrednosti struje armature, poziva se i regulator struje koji na tu referentnu vrednost i reaguje i menja napon armature.

Nedostatak ovog rešenja je u tome što se u PWM prekidima u kojima se pozivaju oba regulatora izvršava značajno više računarskih operacija. Time oni traju duže i mogu dovesti do povremenog zagušenja u radu mikroprocesora, tj. ugrožen je rad u realnom vremenu. Ovo zagušenje se uobičajeno dešava samo povremeno, i teško je za detekciju.



Slika 4.49. Blok dijagram programa PWM prekidne rutine programa za regulaciju brzine JM upravljanjem strujom armature samo sa PWM prekidnom rutinom

4.9.3. Primer mikroprocesorskog C programa za regulaciju brzine JM

U ovom poglavlju su opisana dva mikroprocesorska C programa za regulaciju brzine JM u pogonu sa četvorokvadrantnim čoperom. Programi nisu dati u potpunosti, opisani su samo delovi od interesa za regulaciju brzine JM upravljanjem naponom i strujom, detaljnije u [2_4].

Programi su napisani u skladu sa blok dijagramima algoritma programa opisanim u prethodnim poglavljima, uz određene razlike koje pojednostavljaju račun potrebnog faktora ispunje. Radi jednostavnosti, pri računu diskretnih zakona upravljanja je korišćena matematika sa pokretnim zarezom (*floating point*) jer se smatra da program za regulaciju JM nije računski zahtevan. Takođe, korišćen je standardni C programski jezik tako da se delovi programa mogu preneti i na druge mikrokontrolere. Potpuna prenosivost kod mikroprocesorski *embedded* sistema (sistema koji su namenjeni upravljanju specijalnim hardverom) nije moguća jer napisan C kôd često zavisi od korišćenog hardvera i načina na koje treba čitati/pisati registre periferija koji se razlikuju za različite mikrokontrolere/DSP. U datim primerima, kada je to bilo neophodno, C program je pisan za Microchip DSP seriju dsPIC30F4011.

4.9.3.1. Opis pojedinih delova C programa

Program u C jeziku počinje da se izvršava pozivom *main()* funkcije. Funkcija sa ovakvim imenom uvek mora da postoji i u njen početak treba postaviti deo programa koji želimo da se izvrši odmah nakon resetiranja mikrokontrolera.

Program koji se poziva pre C programa - *Startup* kod

Važno je razumeti da je C viši programski jezik i da za pravilno izvršavanje zahteva određena podešavanja. Iz tog razloga kompajler automatski, na mesto reset vektora linkuje asemblerski *startup* kôd, koji se prvi izvršava nakon resetiranja, podešava sve potrebne uslove za rad C programa i tek onda poziva funkciju *main()*. *Startup* kôd je neophodan, zato njega obezbeđuje proizvođač mikrokontrolera i generiše se automatski, ali programer se uobičajeno njime ne mora baviti. Njegova funkcija je višestruka; npr. ako C program radi sa neinicijalizovanim (*int var1;*) i inicijalizovanim promenljivama (*int var2=100;*) za njih linker obezbedi dve 16-bitne lokacije u RAM. Ali, C standard podrazumeva i da je u tim lokacijama inicijalno upisano *var1 = 0* i *var2 = 100*, u protivnom, može doći do greške u programu. Pošto je sadržaj RAM nepoznat nakon dolaska napajanja, *startup* kôd sadrži dve programske petlje, prva postavlja sve neinicijalizovane RAM lokacije na nula, dok druga kopira sve inicijalizovane vrednosti iz FLASH u RAM. *Startup* kôd podešava i stek pointer, koji C program koristi za prenošenje argumenata funkcijama, itd. Na kraju asemblerski *startup* program poziva *main()* funkciju, asemblerskom naredbom *lcall _main*. Nakon ovog poziva C program preuzima kontrolu i pošto ostaje u beskonačnoj petlji, nikad tu kontrolu više ne ispušta.

Deo C programa kojim se vrši inicijalizacija periferija

Ovaj deo C koda se izvršava samo jednom, nakon resetiranja. U njemu je neophodno podesiti rad CPU i svih potrebnih periferija. U ovom delu kôd poziva potprograme za:

1. Početno podešavanje (inicijalizaciju) sistema, na primer sistemski *clock*, *memory clock*, *wait* stanja u radu sa memorijom, startovanje *watchdog timer*-a, itd,
2. Početno podešavanje (inicijalizaciju) perifernih modula. U slučaju regulacije brzine JM treba nam A/D, PWM i brojački modul, i UART modul za serijski prenos podataka,
3. Inicijalizaciju globalnih promenljivih, pojačanja regulatora, zadatih vrednosti, itd,
4. Na kraju, dozvoljavaju se potrebni prekidi, prekid Timer1 i PWM prekid (samo za strujnu regulaciju), i ulazi se u drugi deo *main* programa, beskonačnu petlju (*mainloop*).

Funkcije za podešavanje pojedinih periferija uobičajeno daje proizvođač i nalaze se u posebnim datotekama. Svaka periferija ima svoju rutinu za podešavanje. U zavisnosti od toga koja nam periferija treba, u toku inicijalizacije pozivamo njenu funkciju za inicijalizaciju, kao na primer PWMinit(), TimerInit(), ADCinit(), QEInit(), UARTinit. Funkcije za inicijalizaciju pojedinih periferija se razlikuju za razne kontrolere i ovaj deo C program uglavnom nije prenosiv. Kao primer, data je C funkcija za inicijalizaciju PWM modula u sklopu dsPIC30f4011:

```
#include <pwm.h>
unsigned int Period = 1500;    // odredjivanje PWM frekvencije, 20kHz
void PWM_Init(void)
{
    ConfigIntMCPWM(PWM_INT_EN & PWM_INT_PR1 & PWM_FLTA_DIS_INT);
    SetDCMCPWM(1,Period,1); // PWM 1, pocetni duty na 50% (4-kvc), update enable
    OpenMCPWM(Period,0,PWM_EN&PWM_IDLE_CON&PWM_OP_SCALE1&PWM_IPCLK_SCALE1&
    PWM_MOD_FREE,PWM_MOD1_COMP&PWM_PDIS3H&PWM_PDIS2H&PWM_PEN1H&PWM_PDIS3L&
    PWM_PDIS2L&PWM_PEN1L, PWM_SEVOPS1&PWM_OSYNC_PWM&PWM_UEN);
}
```

Ova funkcija koristi definicije, strukture i funkcije koje su definisane u pwm.h i biblioteci koju je obezbedio Microchip i koje važe samo za dsPIC30F familiju. Funkcija PWM_init() prvo poziva specijalnu funkciju ConfigIntMCPWM za podešavanje dsPIC PWM prekida, zatim SetDCMCPWM za podešavanje PWM kanala i početnog faktora ispune, kao i funkciju OpenMCPWM za podešavanje moda rada, tipa nosioca, tipa PWM izlaza i konačno otvaranje PWM izlaza, detaljnije u [2_4]. Ovu PWM_init() funkciju očigledno nije moguće iskoristiti na ostalim mikrokontrolerima, za te kontrolere je potrebno pronaći ili načiniti sličnu, koja podešava PWM izlaze datog kontrolera, PWM nosioc tipa testera, upisuje željeni PWM period i početni faktor ispune, zabranjuje/dozvoljava spoljašni ulaz za zaštitu, itd.

Glavna petlja u C programu (mainloop)

Glavna petlja počinje da se izvršava nakon inicijalizacije i program nikad iz nje ne izlazi. Izvršenje glavne petlje se automatski privremeno suspenduje po nailasku nekog prekida. Glavna petlja nastavlja tamo gde je stala čim se izvrši za pristigli prekid odgovarajuća prekidna rutina. Glavnu petlju karakteriše while (1) naredba, kojom se ne dozvoljava izlazak iz nje.

```
while (1)
{
    FunkcijaCijeVremeIzvršenjaNijeKriticno();
}
```

U datom primeru C programa osnovna petlja zove tri funkcije, čiji listinzi nisu dati jer nisu u direktnoj vezi sa upravljanjem energetske pretvaračem:

1. Izvršava sporu mašinu stanja koja pamti u kom stanju se pogon nalazi (radi, ne radi, u kvaru, itd.) kao i razlog zašto smo u tom stanju (fatalni kvar, ulaz van opsega, itd.),
2. Prati da li je primljena komanda sa serijske veze ili da li treba neke podatke poslati,
3. Provera stanja raznih lokalnih ulaza i izlaza (tasteri start, stop, itd.).

Prekidne rutine u C programu

U prekidne rutine se linkuje deo C programa koji mikroprocesor treba da izvršava sinhrono, u tačno određenim vremenskim intervalima i/ili trenucima. Većina modula unutar mikrokontrolera imaju mogućnost generisanja prekida ka mikroprocesoru. Ako je taj prekid prilikom inicijalizacije dozvoljen, po njegovom dolasku mikroprocesor prekida izvršenje glavne petlje i prelazi na izvršenje prekidne rutine koja je namenjena toj periferiji. Prekidna rutina jednog prekida je specijalna funkcija koju treba postaviti (linkovati) na mesto u memoriji

na koje pokazuje adresni vektor koji odgovara tom prekidu (tzv. vektor prekida). Takođe, veoma je važno da C kompajler funkcije koje se izvršavaju u prekidnim rutinama tretira drugačije od običnih funkcija koje se pozivaju iz glavnog programa. Razlog za to je što ove funkcije mogu da prekinu izvršenje bilo koje druge operacije ili funkcije u glavnom programu, i potrebno je sačuvati sadržaje registara koje možda prekinute funkcije/operacije koriste, i čija promena bi ugrozila pravilan nastavak glavnog programa. U tu svrhu se i koristi kodna reč *interrupt* koja u deklaraciji funkcije obaveštava C kompajler da se radi o funkciji koja će biti pozivana nakon detektovanja nekog od prekida. C kompajler ovim funkcijama dodaje dodatni kôd u kome pre izvršenja funkcije na stek postavlja sve radne i statusne registre koje ta funkcija koristi, čime čuva njihove sadržaje. Po završetku prekidne funkcije C kompajler postavlja kôd za vraćanje originalnih sadržaja korišćenih registara. Nakon povratka vrednosti svih radnih i statusnih registara sa steka glavni program može pravilno da nastavi tamo gde je stao.

C program za regulaciju brzine JM upravljanjem naponom koristi samo prekid brojačkog modula, na čiji je vektor linkovana funkcija `T1Interrupt()`:

```
void __attribute__((__interrupt__)) _T1Interrupt(void)
{
// ovde napisati program koji na osnovu Brzine_ref i Brzine menja PDC
}
```

C program za regulaciju brzine JM upravljanjem strujom koristi dva prekidna vektora, na koje su linkovane funkcije `T1Interrupt()` i `PWMInterrupt()`:

```
void __attribute__((__interrupt__)) _T1Interrupt(void)
{
// ovde napisati program koji na osnovu Brzine_ref i Brzine menja Ia_ref
}
void __attribute__((__interrupt__)) _PWMInterrupt(void){
// ovde napisati program koji na osnovu Ia_ref i Ia menja PDC
}
```

4.9.3.2. Delimični listing C programa za regulaciju brzine JM upravljanjem naponom

U ovom poglavlju je dat delimični listing C programa za dsPIC30F4011 za regulaciju brzine JM upravljanjem naponom. Dati program nije potpun, detaljnije u [2_4].

```
int main(void)
{
// 1. Inicijalizacija periferija
GPIOInit();           // podesavanje pinova, selekcija periferija ili GPIO,
                       // selekcija ulaz ili izlaz
UARTInit();           // podesavanje parametara UART, obicno zelimo serijskom
                       // vezom da pratimo rad pogona,
PWMinInit();           // podesavanje parametara PWM nosioca i signala
TimerInit();           // podesavanje koriscenog Timer1, prekid svakih 10ms
ADCInit();             // podesavanje broja ADC ulaza, formata rezultata,
                       // nacina odabiranja ...
// 2. Inicijalizacija promenljivih i konstanti regulatora brzine
ref_brzina =0;         // Da se motor ne pokrene po dolasku napajanja
Ref_napon=0;           // Resetuje se suma PI regulatora i resetuje se
SpeedErr_old=0;        // prethodna greska. Ove promenljive je neophodno
                       // resetovati pre svakog starta a ne samo ovde.
// 3.glavna petlja, izvršava se u pozadini
                       // U glavnoj petlji se ceka Timer 1 prekid i vrse
```

```

// asinhrona radnja. U ovom primeru glavna petlja
// zove tri funkcije ciji pocetak i vreme izvršenja
while (1) // nisu kritični. Glavna petlja može biti i prazna,
{
    StateMachine(); // Spor monitoring rada sistema,
    UARTprocess(); // Spora obrada komunikacije,
    IOprocess(); // Rad sa sporim IO, tasteri, display, LED ...
}
// označava kraj main funkcije

```

Delimični listing prekidne rutine Timer 1 za regulaciju brzine JM upravljanjem naponom:

```

// globalne promenljive korisćene u ovoj rutini
float Kiw = 0.1; // parametri regulatora i izlaz su floating
float Kpw = 2.1; // point brojevi. Lakše ali sporije rešenje
float Ref_Napon; // Izlaz regulatora je takodje float broj
int PWMdutyCycle; // Faktor ispune za PDC mora biti integer tipa
int SpeedErr_old; // speed_err upamćena iz prethodnog PWM perioda
float Kw = SPEED_MAX_RPM/1023; // konstanta za konverziju brzine u rpm
// prekidna rutina Timer1 modula
void __attribute__((interrupt)) _T1Interrupt(void)
{
    float brzina; // lokalne promenljive korisćene u prekidnoj rutini
    float Inc; // inkrementalni deo PI regulatora, dat kao lokalna
    // varijabla jer inkrement ne mora da se pamti
// 1. citanje zadate i stvarne (merene) brzine
    Ref_brzina = (float)Kw*ReadADC (0); //prosto (ali rač. najzahtevnije)
    brzina = (float) Kw*ReadADC (1); // je sve odmah prebaciti u float
// 2. PI regulator brzine u ink. formi - racun ref. napona za 4-kvadr. coper
    SpeedErr=Ref_brzina - brzina; // 2a. najnovija vrednost greske
    // 2b. racun Inc dela zakona upravljanja
    Inc = Kiw * SpeedErr + Kpw * (SpeedErr - SpeedErr_old);
    SpeedErr_old= SpeedErr; // 2c. memorisanje stare greske za
    // sledeći period
    Ref_Napon = Ref_Napon + Inc; // 2d. zatim se racuna ukupni izlaz
    // Ref. napon je float promenljiva i
    // sumiranje radi i za male inc vrednosti
// 3. Limitovanje izlaznog napona (faktora ispune) za 4-kvc
// Ref_Napon treba ograniciti na maksimum naponske komande, za 4kvad coper
// možemo dozvoliti i pozitivnu i negativnu naponsku komandu
    if (Ref_Napon > PWM_PERIOD/2) Ref_Napon = PWM_PERIOD/2;
    if (Ref_Napon <= PWM_PERIOD/2) Ref_Napon -= PWM_PERIOD/2;
// 4. Racun faktora ispune - ovde se referentni napon pretvara u faktor
// ispune. Ref. napon 4kvad copera (±UDC) se preslikava u 0- PWM_PERIOD
// Paznja, Ref_Napon je i dalje float, sa nepotrebnim decimalama u kojima //
se pamte mali inkrementi. Sada je vreme za upis u PDC koji je integer
// format (nema decimala) i zato float promenljivu privremeno prebacujemo
// u integer format (ukidamo decimala). To se zove kastovanje, privremena
// (samo u toku tekuće operacije) promena formata. napr: (int)956,26= 956

    PWMdutyCycle =(int) Ref_Napon + PWM_PERIOD/2;
// 5. Upisujemo novi faktor ispune u PDC1
    PDC1=2*PWMdutyCycle; // dva puta veci samo zbog dsPIC PWM modula
// 6. Brisemo Timer interrupt flag
    IFS0bits.T1IF = 0;
}

```

4.9.3.3. Delimični listing C programa za regulaciju brzine JM upravljanjem strujom

Glavni program je sličan onome u pogl. 4.9.3.2, dok je delimičan listing C programa Timer prekidne rutine za upravljanjem strujom armature dat u nastavku:

```
// globalne promenljive koriscene u ovoj rutini
float Kiw = 0.1;           // parametri regulatora brzine
float Kpw= 2.1;
float Kw = SPEED_MAX_RPM/1023; // konstanta za konverziju brzine u rpm
float Ia_Ref;              // Referentna struja armature
int Speed_Err, Speed_Err_old;

void __attribute__((__interrupt__)) _T1Interrupt(void)
{
    float brzina, Inc;      // lokalne promenljive brzina i Inc PI reg
    // citanje zadate brzine
    Ref_brzina = (float) Kw*ReadADC (0);
    // merenje stvarne brzine
    brzina = (float) Kw*ReadADC (1);
    // Regulator brzine
    Speed_Err=Ref_brzina - brzina;
    Inc = Kiw * Speed_Err + Kpw * (Speed_Err - Speed_Err_old);
    Speed_Err_old= Speed_Err;
    Ia_Ref = Ia_Ref + Inc;
    // Limitovanje izlazne struje za 4-kvc
    if (Ia_Ref > I_MAX) Ia_Ref = I_MAX;
    if (Ia_Ref < - I_MAX) Ia_Ref = - I_MAX;
    // Brisanje Timer interrupt flag
    IFS0bits.T1IF = 0;
}
```

Konačno, sledi listing C programa PWM prekidne rutine za upravljanje strujom:

```
float Kii = 1;             // parametri regulatora struje
float Kpi= 20;
float Ki = I_MAX_A/1023; // konstanta za konverziju struje u A
float Ia;                 // merena struja u A
float Ref_Napon;          // Izlaz regulatora je takodje float broj
int PWMdutyCycle;        // Faktor ispune
int I_err, I_err_old;

void __attribute__((__interrupt__)) _PWMInterrupt(void)
{
    float Inc;             // Inc Pi regulatora moze biti lokalna promenljiva
    Ia = (float)Ki*ReadADC (2);
    I_Err=Ia_ref - Ia;
    Inc = Kii * I_Err + Kpi * (I_Err - I_Err_old);
    I_Err_old= I_Err;

    Ref_Napon = Ref_Napon + Inc;
    // limit faktora ispune za 4-kvc
    if (Ref_Napon > PWM_PERIOD/2) Ref_Napon = PWM_PERIOD/2;
    if (Ref_Napon <= PWM_PERIOD/2) Ref_Napon -= PWM_PERIOD/2;

    PWMdutyCycle =(int) Ref_Napon + PWM_PERIOD/2;
    PDC1=2*PWMdutyCycle;   // dva puta veci samo zbog dsPIC PWM modula

    IFS2bits.PWMIF = 0;    // dozvoli ulazak u PWM prekid i sledeci put
}
```