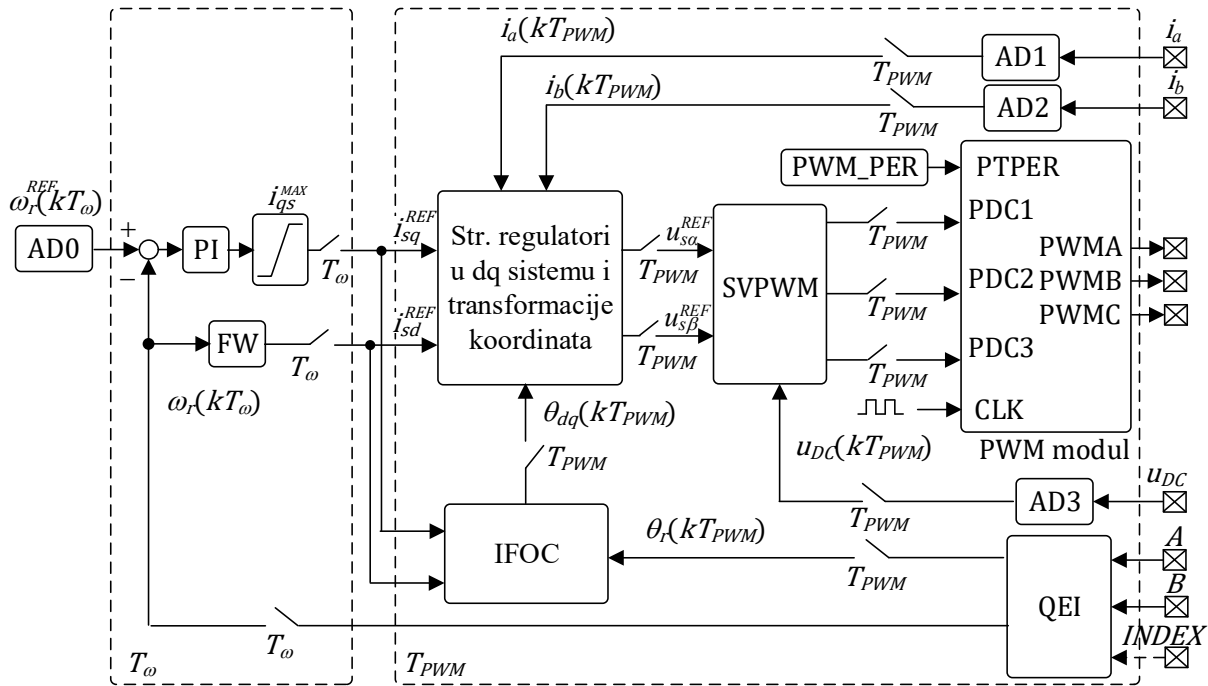


6.13. Mikroprocesorski program za indirektno vektorsko upravljanje i regulaciju brzine asinhronog motora

Mikroprocesorski program za indirektno vektorsko upravljanje treba da upravlja momentom AM tako da se ostvari referentna brzina rotora, uz fluks rotora koji se drži na nominalnom nivou u I zoni, a smanjuje sa brzinom u II zoni. Potrebni blokovi za mikrokontrolersku realizaciju regulacije brzine i IFOC upravljanja AM dati su na slici 6.55.



Slika 6.55. Blokovi mikroprocesorskog prog. za regulaciju brzine i vektorsko upravljanje AM

Samo manji deo programskih blokova mikroprocesorskog programa na slici 6.55 vrši regulaciju brzine AM i slabljenje polja (blok FW, *field weakening*), čiji su rezultat nove referentne vrednosti momenta ($\sim i_{sq}^{REF}$) i fluksa rotora ($\sim i_{sd}^{REF}$). Ovi programski blokovi se uobičajeno izvršavaju u dužoj periodi odabiranja brzine T_ω i na slici 6.55 su grupisani levo:

1. čitanje referentne brzine, u datom primeru čitanje A/D rezultata,
2. merenje brzine, tj. u slučaju enkodera njen proračun na osnovu promene položaja u periodu T_ω . QEI periferija prima impulse sa enkodera i meri promenu položaja, bez opterećenja procesora, koji samo jednom u periodu T_ω treba da pročita rezultat,
3. proračun linearnog PI regulatora brzine koji na osnovu greške brzine upravlja referentnom q komponentnom struje statora i_{sq}^{REF} ,
4. implementacija slabljenja polja koja na osnovu trenutne brzine rotora zadaje referentnu d komponentu struje statora i_{sd}^{REF} .

Ovim akcijama se jednom u periodu T_ω dobijaju vrednosti i_{sd}^{REF} , i_{sq}^{REF} , na osnovu kojih vektorski kontroler treba da ostvari potreban nivo fluksa rotora i potreban moment.

Veći deo mikroprocesorskog programa se bavi vektorskim upravljanjem AM i da bi bio u mogućnosti da prati promenu ugla vektora fluksa rotora treba da se izvršava najbrže moguće,

uobičajeno sa PWM periodom T_{PWM} . Vektorsko upravljanje AM je računski zahtevnije u odnosu na skalarno upravljanje, i samim tim je teže za mikrokontrolersku realizaciju u realnom vremenu. Pošto je PWM perioda relativno kratka, izvršenje ovog dela programa može značajno da optereti mikroprocesor. Ove akcije su na slici 6.55 grupisane desno:

1. Merenje i skaliranje veličina koje je potrebno pratiti svake PWM periode:
 - merenje dve linijske struje AM (i_a, i_b), i račun treće (i_c) fazne struje,
 - merenja napona jednosmernog međukola (u_{DC}),
 - čitanje i skaliranje najnovijeg položaja rotora (θ_r) dobijenog iz QEI periferije.
2. Račun strujnih regulatora u dq sistemu i potrebnih transformacija:
 - transformacija struja $abc/\alpha\beta$ i $\alpha\beta/dq$, rezultat i_{sd}, i_{sq} ,
 - račun dva diskretna PI regulatora struje u dq domenu, rezultat $u_{sd}^{REF}, u_{sq}^{REF}$,
 - transformacija referentnih napona $dq/\alpha\beta$, rezultat $u_{s\alpha}^{REF}, u_{s\beta}^{REF}$.
3. Račun klizanja, tj. indirektni račun položaja fluksa rotora i postavljanje dq sistema:
 - račun ugla klizanja na osnovu dq struja i parametra T_r , rezultat θ_{kl} ,
 - račun ugla dq sistema slaganjem brzina, rezultat je pomeranje polja sa pomeranjem rotora i prednjačenje za potrebno klizanje $\theta_{dq} = \theta_r + \theta_{kl}$.
4. SVPWM račun na osnovu $u_{s\alpha}^{REF}, u_{s\beta}^{REF}$ i upis u PDC registre PWM modula.

6.13.1. Uprošćen blok dijagrama algoritma mikroprocesorskog programa za vektorsko upravljanje AM koji je regulisan po brzini

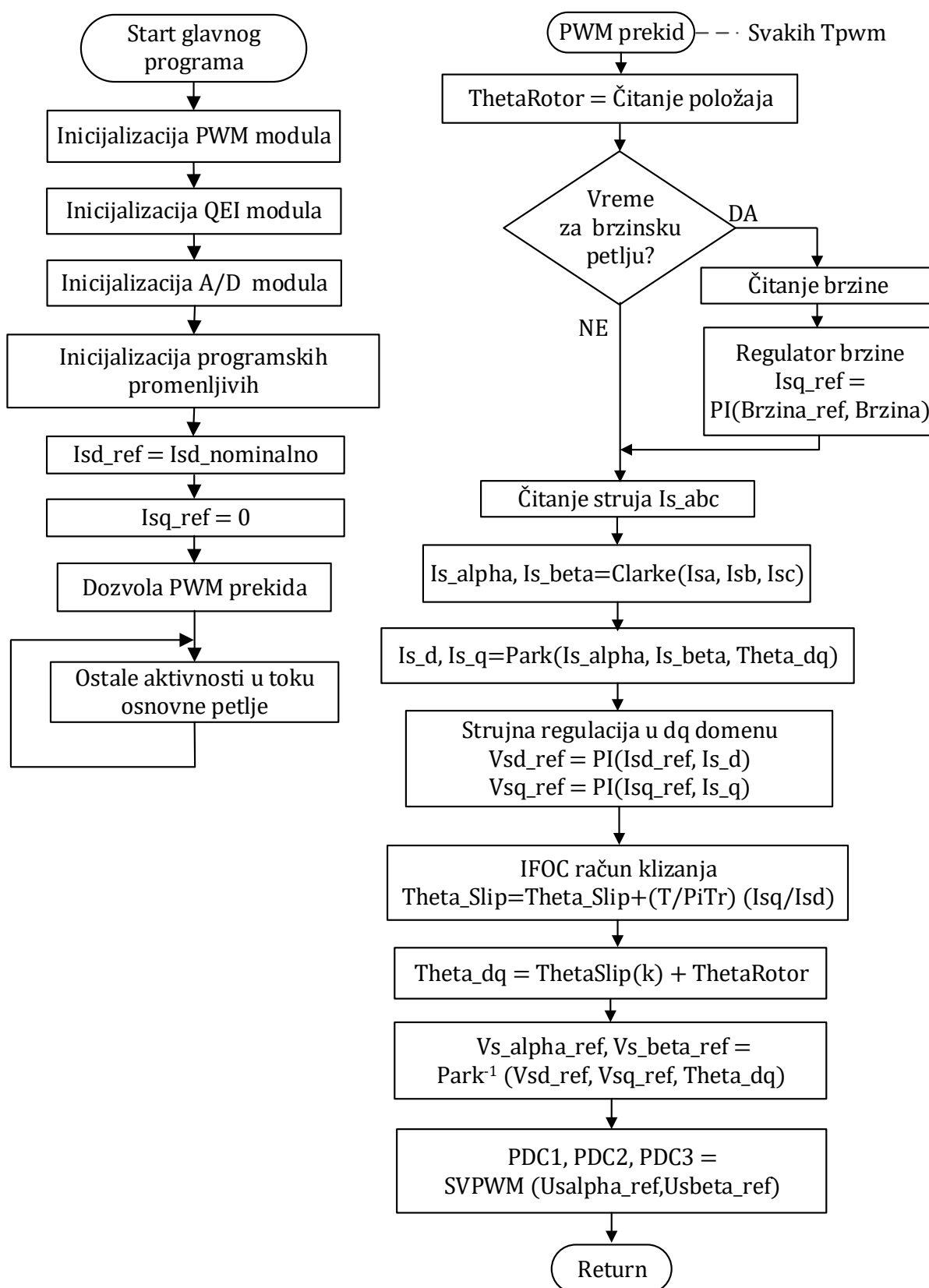
Uprošćen blok dijagram algoritma programa za indirektno vektorsko upravljanje (IFOC) i regulaciju brzine AM je dat na slici 6.56. Akcenat je stavljen na jednostavnost, i date su osnovne akcije, mesto i redosled njihovog pozivanja, bez detaljnog opisa.

Blok dijagram glavnog programa je dat na slici 6.56 levo i on se sastoji iz dela za inicijalizaciju i glavne programske petlje. U delu za inicijalizaciju se vrši: 1. podešavanje periferija (PWM modula, A/D modula i QEI modula), 2. postavljanje programskih konstanti (parametri IFOC kalkulatora, pojačanja regulatora struje i brzine) i 3. podešavanje globalnih promenljivih (referentna vrednost d struje postavlja na nominalnu vrednost, i ona se neće menjati tokom programa (rad samo u osnovnom opsegu brzina), referentna vrednost q struje se postavlja na nulu i time se resetuje regulatora brzine, itd.). Ovaj primer programa zatim ulazi u beskonačnu glavnu petlju u kojoj nema operacija i samo se čeka dolazak novog PWM prekida.

Blok dijagram algoritma PWM prekidne rutine koja treba da se izvršava na početku svakog PWM perioda je dat na slici 6.56 desno.

Jednom u T_ω/T_{PWM} ulazaka u ovaj prekid, kada brojač detektuje da je prošla perioda T_ω , u ovom posebnom delu PWM rutine se vrši digitalni zakon upravljanja brzinom. Rezultat ovog dela programa je nova zadata struja q ose (Isq_ref), koja se ne menja sledećih T_ω/T_{PWM} PWM perioda, i zajedno sa nominalnom d strujom (Isd_ref) definiše novi referentni strujni vektor.

U svakom PWM prekidu se na osnovu referentnih dq koordinata struje (Isd_ref, Isq_ref) vrši strujna regulacija (odabiranje i transformacija merenih struja u dq domen i račun dva PI regulatora), IFOC račun klizanja i pomeranje ugla dq sistema, i transformacija referentnih dq napona (Vsd_ref, Vsq_ref) nazad u $\alpha\beta$ domen ($Vs_alpha_ref, Vs_beta_ref$) gde se one koriste kao ulaz SVPWM koji računa nove faktore ispune. Ovim se najbrže moguće u vremenu prati pomeranje vektora fluksa rotora i u realnom vremenu pomera vektor napona, čime se u svakom trenutku utiskuje vektor struje potreban da se ostvare zadati fluks i moment AM.



Slika 6.56. Uprošćen blok dijagram mikroprocesorskog programa za IFOC upravljanje i regulaciju brzine AM

6.13.2. Detaljniji blok dijagrama algoritma mikroprocesorskog programa za vektorsko upravljanje AM koji je regulisan po brzini

U ovom poglavlju je dat detaljniji blok dijagram mikroprocesorskog programa vektorski upravljanog pogona sa AM sa indirektnom procenom položaja vektora fluksa rotora (*IFOC*) i zatvorenom povratnom spregom po brzini rotora, slike 6.57 – 6.61.

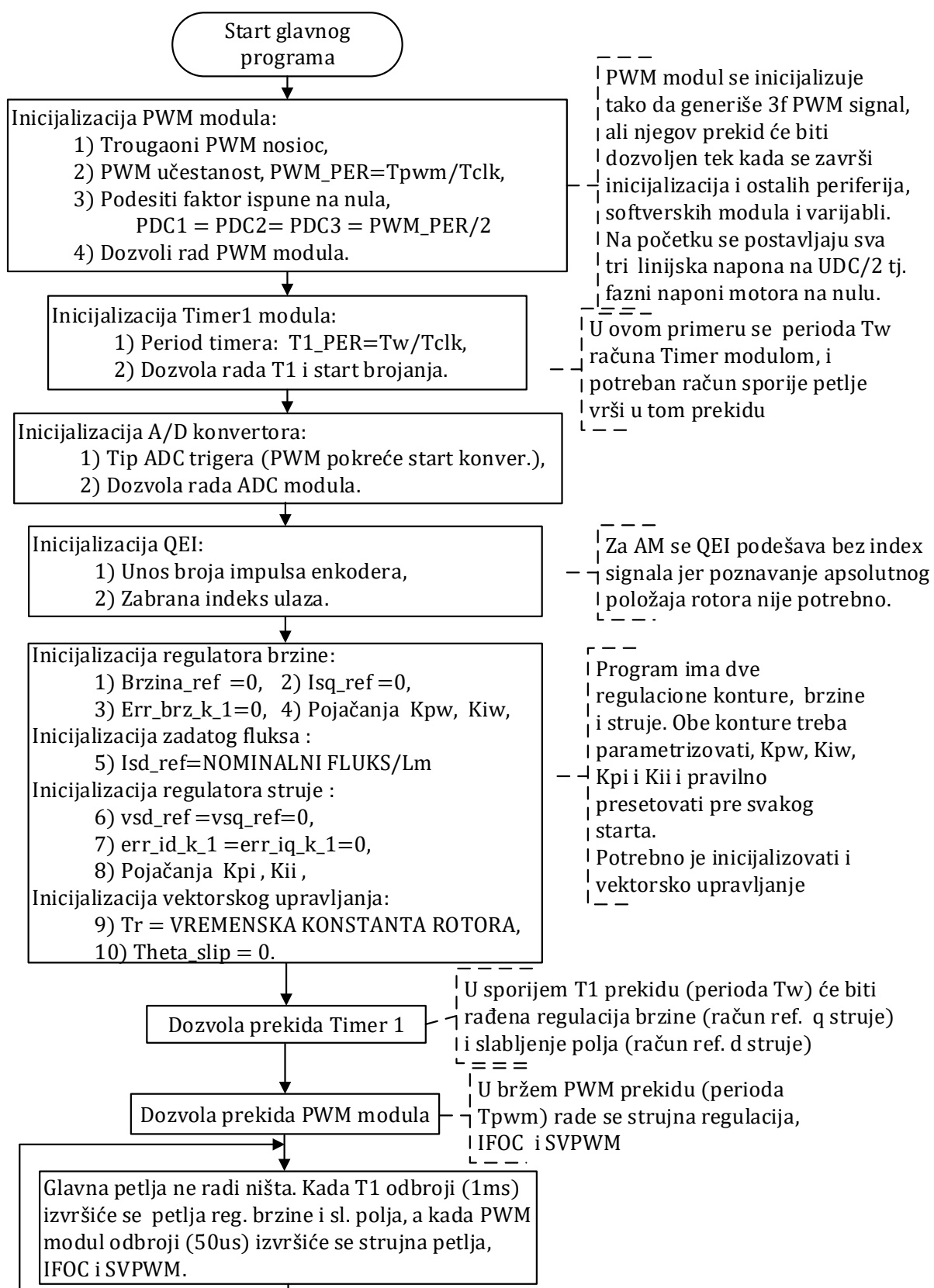
Na slici 6.57 je dat algoritam glavnog programa koji se pokreće nakon reseta i u kome su detaljnije opisane sledeće radnje: 1. inicijalizacija periferija (PWM, A/D konvertor, Timer i QEI modula), 2. inicijalizacija programskih konstanti (parametara *IFOC* kalkulatora, regulatora struje i brzine), i 3. inicijalizacija programskih promenljivih (suma na izlazu regulatora brzine i sume na izlazima dva regulatora struje se postavljaju na nulu, itd.). Nakon toga, glavni program ulazi u beskonačnu petlju u kojoj čeka prekid brojačkog (Timer) modula (signal da je prošla perioda T_ω) ili prekid PWM modula (signal da je prošla perioda T_{PWM}).

Na slici 6.58 je dat algoritam prekidne rutine Timer modula koji se izvršava svake periode T_ω i u kome se pozivaju PI regulator brzine i račun slabljenja polja. Regulator brzine na osnovu razlike između referentne i izmerene brzine upravlja referentnom vrednošću q komponente struje statora. U osnovnom opsegu brzina referentna vrednost q struje se limitira na osnovu zadate vrednosti d struje i strujnih mogućnosti invertora ($i_{sq}^{MAX} = \sqrt{i_{MAX}^2 - i_{sd}^{REF}}$), na ovaj način se u strujnom limitu praktično zadržava željeni fluks, a ograničava moment. Ali, na velikim brzinama obrtanja, sa padom fluksa, pri svakom porastu momenta klizanje značajno raste i potrebno ga je ograničiti. Ovo je u *IFOC* pogonu najlakše ostvariti dodatnim limitiranjem referentne vrednosti q struje na osnovu $\omega_{kl} = (1/T_r)(i_{sq}/i_{sd})$, primenom $i_{sq}^{MAX_KL} = T_r i_{sd}^{REF} \omega_{kl}^{MAX}$. Duboko u slabljenju polja vrednost d struje značajno opada i limit q struje usled ograničenja klizanja $i_{sq}^{MAX_KL}$ postaje sve manji i sve dominantniji. Blok za slabljenje polja održava referentnu vrednost d struje statora na nominalnoj vrednosti sve do nominalne brzine, a zatim je proporcionalno smanjuje sa daljim porastom brzine. Nakon završetka ove prekidne rutine, izračunate su nove referentne vrednosti dq komponenti struje (i_{sd_ref} , i_{sq_ref}), koje neće biti promenjene do sledećeg prekida Timer modula, odnosno do isteka periode T_ω .

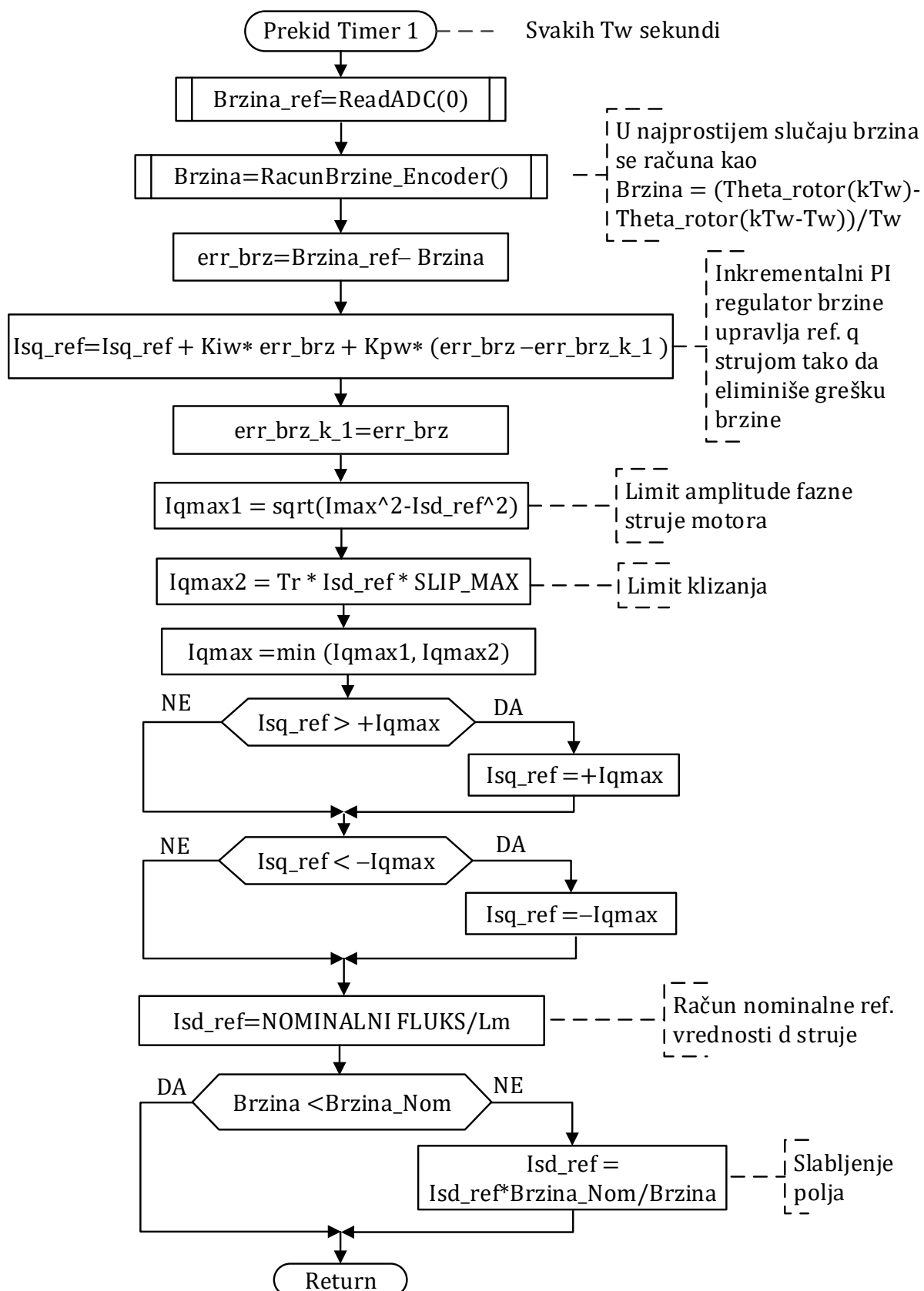
Na slikama 6.59 – 6.61 su dati delovi blok dijagrama algoritma PWM prekidne rutine programa za *IFOC* upravljanje i regulaciju brzine AM. Prekidna rutina počinje da se izvršava kada PWM modul signalizira početak novog PWM perioda i u njoj su realizovana potrebna merenja, strujna regulacija u dq sistemu osa, *IFOC* kalkulator klizanja i SVPWM račun.

U prvom delu PWM prekidne rutine prikazanom na slici 6.59 se vrše merenja dve linijske struje, napona jednosmernog međukola i merenje trenutnog položaja rotora. Zatim se vrše $abc/\alpha\beta$ i $\alpha\beta/dq$ transformacije struja kojima se dobijaju najnovije merene vrednosti struja (i_{sd} , i_{sq}). Konačno, kao priprema za regulatore struje se na osnovu izmerenog napona jednosmernog međukola V_{dc} računaju i limiti napona (V_{sd_limit} , V_{sq_limit}). Na slici je prikazano rešenje u kojem je limit napona $V_{dc}/\sqrt{3}$, ali ukoliko je potrebno više napona naponski limit je moguće računati i na osnovu $2V_{dc}/3$ (dozvoljen i nelinearni režim SVPWM).

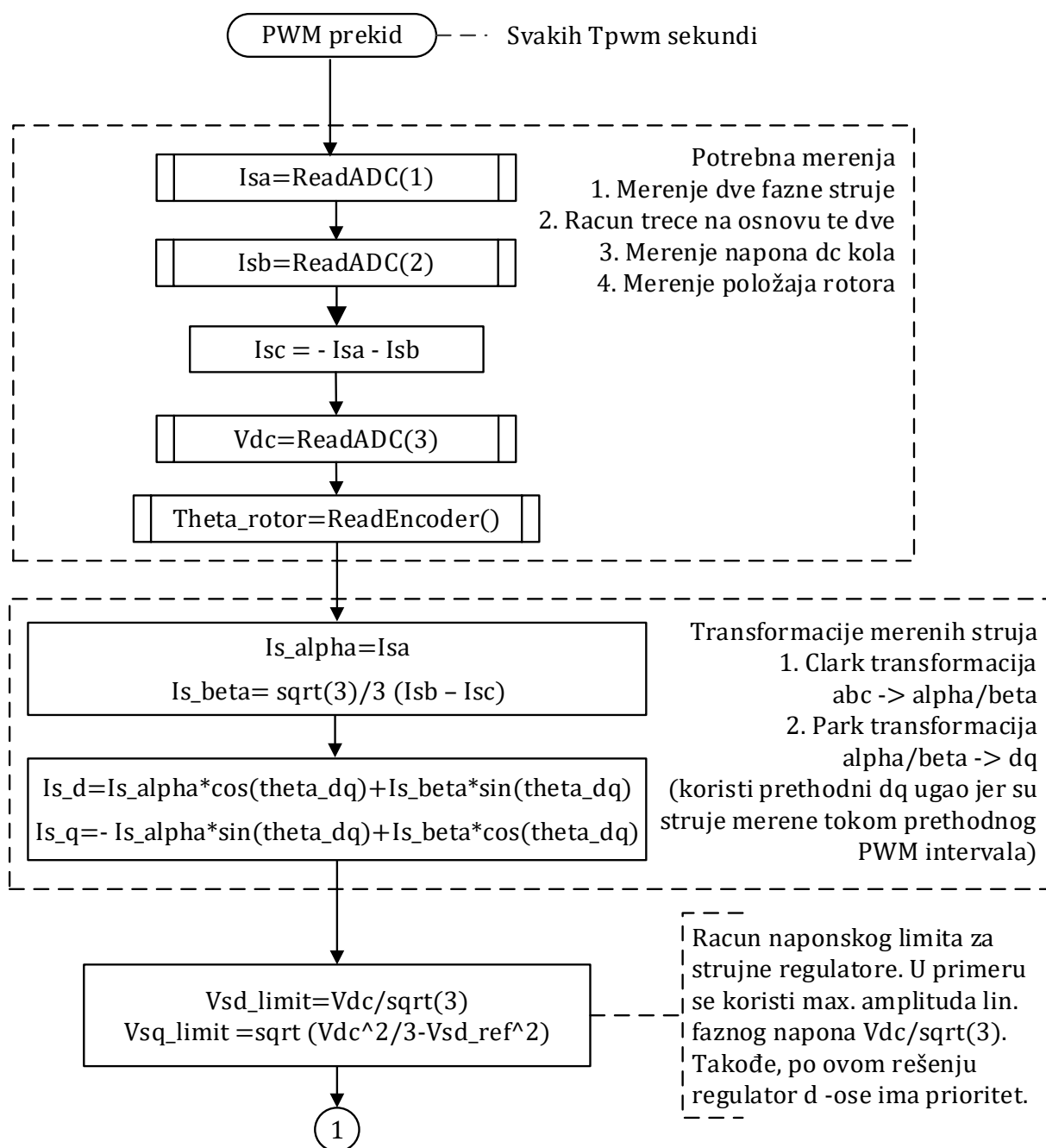
U drugom delu PWM prekidne rutine, slika 6.60, se prvo izvršavaju dva diskretna PI regulatora struje sa odgovarajućim limitovanjem izlaznih napona. Zatim se vrši *IFOC* račun novog ugla klizanja (Θ_{slip}) koji se dalje sabira sa novoizmerenim uglom rotora (Θ_{rotor}), i time se vrši pomeranje dq koordinatnog sistema (Θ_{dq}). Na ovaj način se vektor referentnog izlaznog napona pomera sa potrebnim ugaonim inkrementima u vremenu u svakoj novoj PWM periodi, i time u realnom vremenu prati pomeranje vektora fluksa rotora.



Slika 6.57. Blok dijagram algoritma IFOC programa: glavni program [4_4]

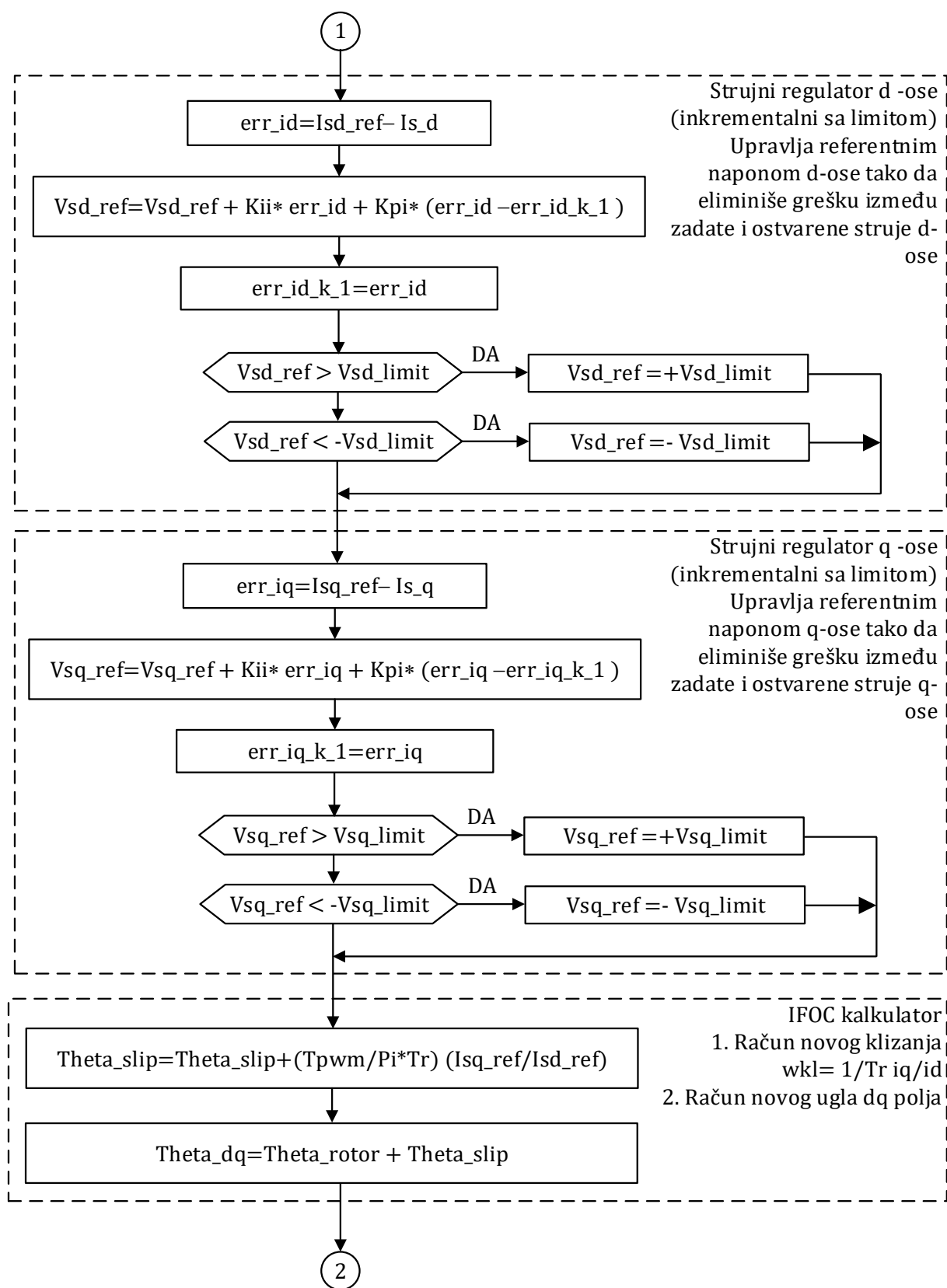


Slika 6.58. Blok dijagram algoritma IFOC programa: brzinska petlja i slabljenje polja [4_4]

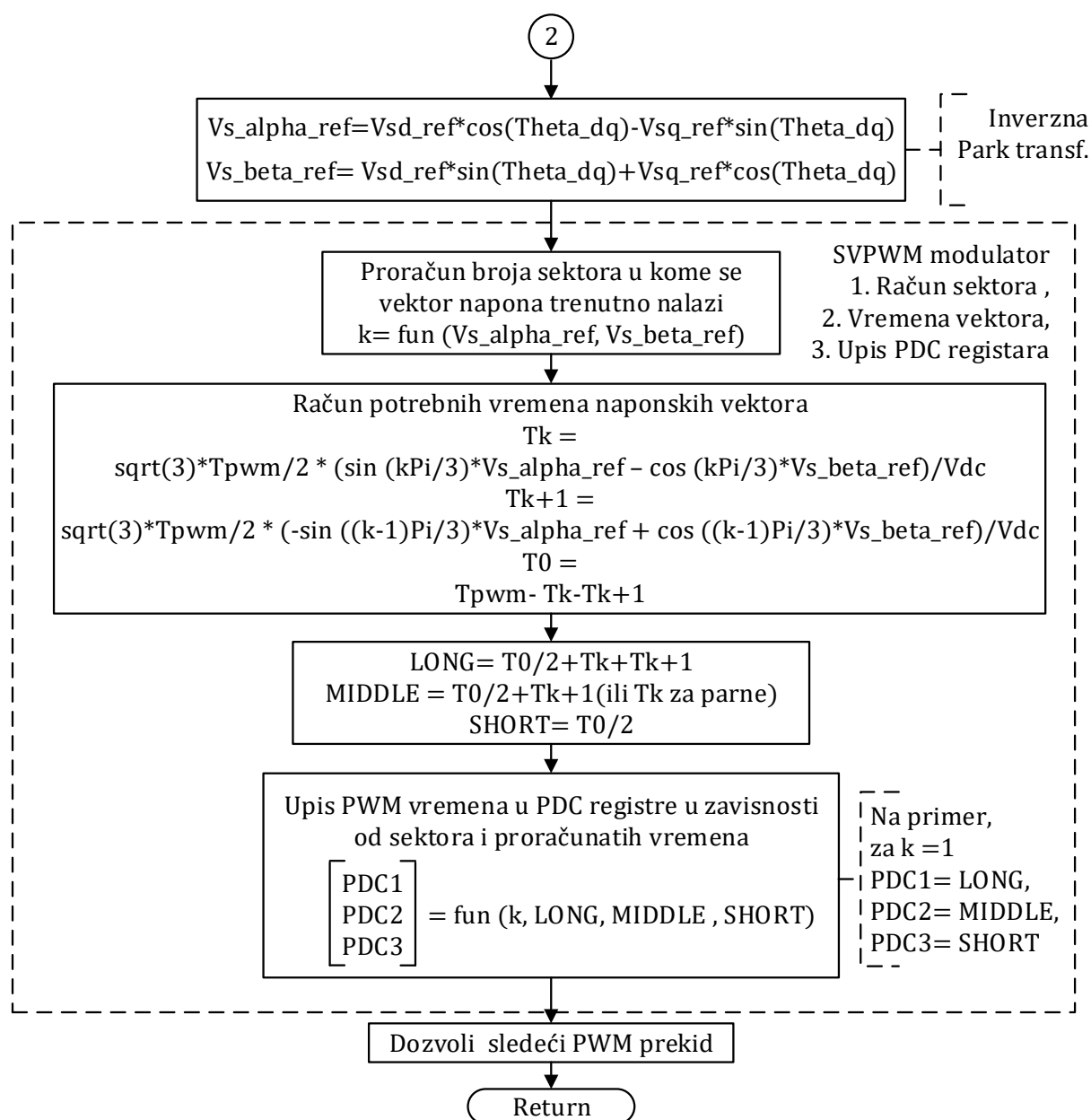


Slika 6.59. Blok dijagram algoritma IFOC programa: PWM prekidna rutina– I deo [4_4]

Konačno, treći deo PWM prekidne rutina dat je na slici 6.61. Algoritam dalje nastavlja sa $dq/\alpha\beta$ transformacijom referentnih napona koji su upravo izračunati kao izlazi strujnih PI regulatora. U transformaciji se koristi nova vrednost ugla dq sistema (Θ_{dq} izračunata u tekućoj periodi), što je i korektno, jer će taj napon usled kašnjenja PWM modula biti realizovan tek u sledećoj PWM periodi. Ovim su izračunati naponi $V_{s_alpha_ref}$ i $V_{s_beta_ref}$ što su potrebne reference za SVPWM račun koji se poslednji poziva. SVPWM računa sektor u kome se referentni napon nalazi, potrebna vremena trajanja aktivnih vektora na osnovu referentnih napona i ulaznog jednosmernog napona V_{dc} , i konačno upisuje potrebna vremena vođenja u PDC registre PWM modula za faktore ispunje.



Slika 6.60. Blok dijagram algoritma IFOC programa: PWM prekidna rutina– II deo [4_4]

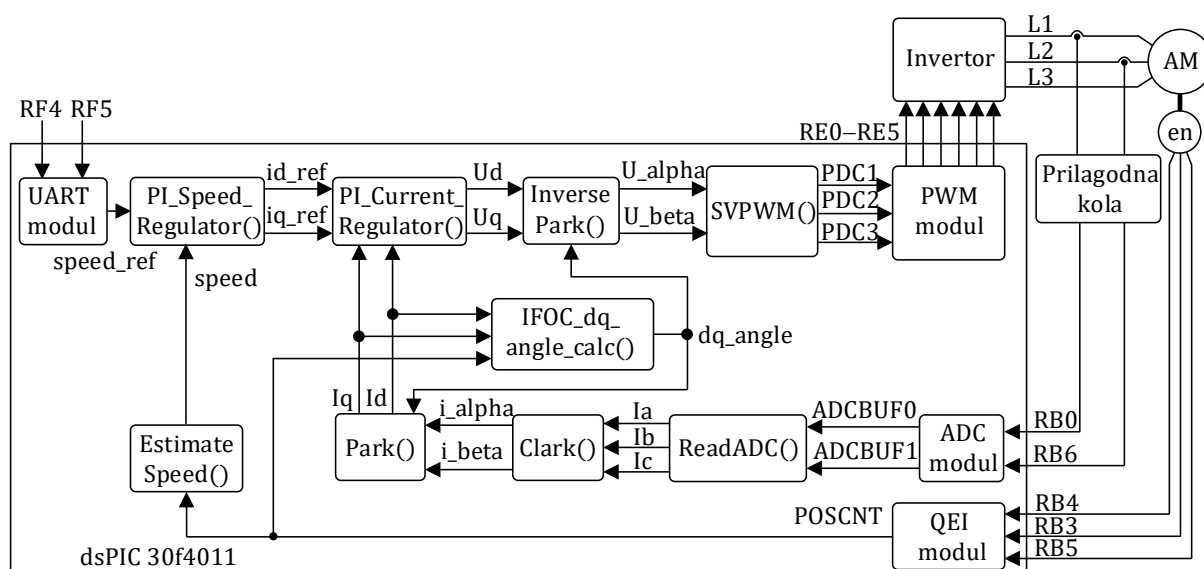


Slika 6.61. Blok dijagram algoritma IFOC programa: PWM prekidna rutina– III deo [4_4]

6.13.3. Primer mikroprocesorskog C programa za dsPIC30F4011

U ovom poglavlju dat je primer realizacije mikroprocesorskog programa koji vrši vektorsko upravljanje i regulaciju brzine AM na Microchip DSP dsPIC30F4011, [6_24]. dsPIC30F4011 je 16-bitni mikrokontroler sa delimičnom Harvard arhitekturom koji radi na *clock* frekvenciji od 30MIPS (*Million Instructions Per Second*) i sadrži sve potrebne module za vektorsko upravljanje: AD modul sa 8 ulaza, PWM modul sa 6 izlaza, QE1 modul za rad sa enkoderom, UART modul za serijsku komunikaciju i mnoge druge.

Na slici 6.62. su označeni upotrebljeni pinovi i periferije mikrokontrolera kao i programski moduli koje koristi C program. Za svaku funkciju su označeni nazivi ulaznih i izlaznih promenljivih, kao i nazivi registara koji sadrže rezultate rada perifernih modula.



Slika 6.62. Pinovi dsPIC30F4011, periferni (HW) i programski (SW) moduli koje koristi program za upravljanje brzinom i indirektno vektorsko upravljanje AM

Raspored poziva pojedinih funkcija programa naznačen je u blok dijagramu algoritma programa koji je prikazan na slici 6.63. Glavni program (levo) nakon reseta vrši: 1. inicijalizacija periferija (PWM, ADC, QEI i UART modula), 2. programskih konstanti (parametri IFOC, regulatora struje i brzine), i 3. programskih promenljivih (d struja na nominalnu vrednost, q struja i zadata brzina na nulu, itd.). Zatim glavni program ulazi u petlju u kojoj proverava da li je stigla nova referentna vrednost brzine rotora sa serijske veze, ako jeste preuzima je, skalira i upisuje u promenljivu *speed_ref* koju koristi regulator brzine. Prekidna rutina (desno) se izvršava svakog PWM perioda (u datom primeru $T_{PWM}=1/12\text{kHz}$) dok se jedan njen deo izvršava samo jednom u 10ms (jednom u 1200 ulazaka). U tom delu se vrši merenje i regulacija brzine. Rezultat izvršenja ovog dela programa je nova zadata struja q ose, koja važi sledećih 10ms. Ona time, zajedno sa nominalnom d strujom, postaje praktično konstanta referenca za dq strujne regulatore koji se izvršavaju najbrže moguće, u svakom PWM periodu. Prvo se mere i računaju (Clarke i Park) tekuće dq koordinate strujnog vektora, zatim strujni regulatori koriguju naponske dq komande za sledeći PWM period tako da se izmerene dq struje što pre izjednače sa zadatima. Novi dq ugao se računa na osnovu novog ugla rotora i ugla klizanja i koristi za $dq/\alpha\beta$ transformaciju novih dq napona. Dobijene $\alpha\beta$ koordinate naponskog vektora su ulaz SVPWM modula koji generiše potrebne faktore ispunje.

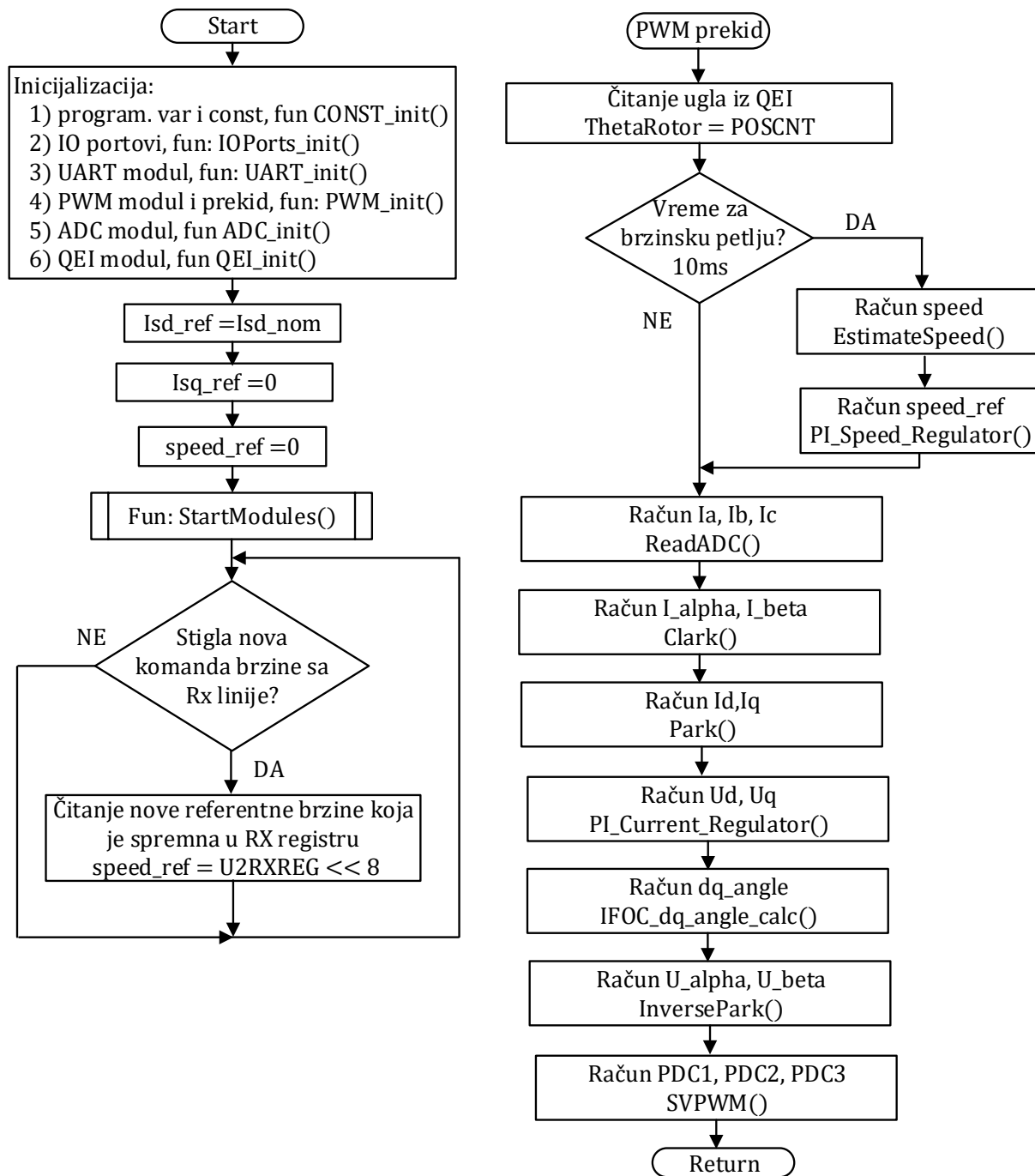
U daljem tekstu ukratko su objašnjene sve konstante i promenljive, svi moduli C programa, funkcije koje C program poziva, njihov kratak opis i listing C programa.

6.13.3.1. Globalne programske promenljive i konstante

U ovom poglavlju je dat opis konstanti i globalnih promenljivih koje koristi program.

Konstanta *FCY* definiše *clock* frekvenciju sa kojom dsPIC radi, u ovom slučaju 30MHz. Konstanta *FPWM* definiše PWM frekvenciju koja je izabrana da bude 12kHz.

Konstanta *FREQ_SPEED_REG* definiše frekvenciju odabiranja regulatora brzine, podešeno na 100Hz. Konstanta *encoder_inc* je broj impulsa enkodera po obrtaju, korišćeni enkoder ima 2500 imp/obr. Konstanta *BASE_SPEED* definiše baznu brzinu rotora, upisano 1500 o/min pošto ovaj primer programa ne predviđa rad iznad nominalne brzine.



Slika 6.63. Blok dijagram algoritma datog C programa za indirektno vektorsko upravljanje AM bazirano na dsPIC30F4011

Konstante K_{p_i} i K_{i_i} su pojačanja PI regulatora struje, upisane vrednosti su dobijene na osnovu proračuna datog u pogl. 6.8.3. Konstante K_{p_speed} i K_{i_speed} su pojačanja regulatora brzine, upisane vrednosti dobijene po pogl. 6.10.2.

Konstante L_r i R_r definišu apsolutne vrednosti otpornosti i induktivnosti rotora. One se koriste samo za izračunavanje vremenske konstante rotora, i mogu se upisati kao *float* brojevi.

Zatim slede izvedene konstante koje se računaju u funkciji *CONST_Init()*. To su početne vrednosti parametara koji se po potrebi mogu menjati u toku izvršenja programa.

Na kraju su definisane sve globalne promenljive koje se koriste u radu programa. Globalne promenljive se pamte po izlasku iz funkcije koja ih izračunava i mogu se koristiti kao prethodna vrednost te iste promenljive pri sledećem računu.

```
// 1. Konstante koje definisu parametri pogona
#define FCY 30000000 // Hz
#define FPWM 12000 // Hz
#define FREQ_SPEED_REG 100 // Hz
#define ENCODER_INC 2500
#define Kp_i 6175 // P dejstvo str. reg, q1.15
#define Ki_i 470 // I dejstvo str. reg, q1.15
#define Kp_speed 16039 // P dejstvo brz. reg, q6.9
#define Ki_speed 2779 // I dejstvo brz reg, q6.9
// 2. Konstante koje definisu parametre motora
#define BASE_SPEED 1500 // RPM
#define Rr 1.05 // Ohm
#define Lr 0.0769 // H
// 3. Koriscenje matematicke konstante
#define ONE_OVER_SQRT_3 0x49E6 // 1/sqrt(3) u q1.15 formatu
#define M_PI 3.141592
// 4. Izvedene konstante koje se racunaju u funkciji CONST_init()
int PWM_PERIOD, HALF_PWM_PERIOD;
int Tpwm_over_Tr,
int One_minus_Tpwm_over_Tr;
long Tpwm_over_TrPi_32b;
int SPEED_REG_PRESCAL, SPEED_CONST, POSITION_CONST;
int BAUD_RATE_GEN;
// Globalne programske varijable koje vide svi prog. moduli
long dq_angle, slip_angle; // dq angle i slip angle u Q1.31
int dq_angle_q15; // dq angle u Q1.15
int SIN_dqAngle; // sin koji odgovara uglu dq sistema
int COSINE_dqAngle; // cosine koji odgovara ugla dq sistema
int U_alpha, U_beta;
int Ud, Uq;
int big, middle, small;
int Ia, Ib, Ic;
int I_alpha, I_beta;
int Id, Iq;
int Id_last, Iq_last, Id_filt_last;
int Id_ref, Iq_ref;
int UQ_LIMIT;
int interrupt_counter;
int speed_ref, speed, speed_last;
int POSCNT_last;
unsigned char sector;
```

6.13.3.2. Funkcija za inicijalizaciju korišćenih programskih konstanti

Program za vektorsko upravljanje AM koristi veliki broj konstanti koje se izvede na osnovu parametara pogona. One se proračunavaju jednom nakon inicijalizacije u funkciji *CONST_Init()*. Isti rezultat u izvedenim konstantama je moguće dobiti i preprocesorskim naredbama, naredbama koje se izvršavaju samo jednom tokom prevođenja (kompajliranja) programa. Ipak, na ovaj način imamo C funkciju koja se može po potrebi pozivati više puta. Pošto se ova funkcija poziva ili samo jednom ili sporom periodom, korišćenje spore *floating-point* matematike je dozvoljeno. Time se uprošćava proračun ali je i dalje važno da je krajnji rezultat prilagođen *fixed-point* matematici koja tu konstantu koristi u drugim funkcijama.

```
inline void CONST_Init()
{
    PWM_PERIOD = (int)((float)FCY / FPWM); // Broj clock ciklusa jednak PWM periodu
    HALF_PWM_PERIOD = (int)((float)FCY / FPWM / 2); // PWM poluperioda, asimet. PWM
```

```

float Tr = (float)Lr / Rr; // vremenska konstanta rotora [s]
Tpwm_over_Tr = (int)(1 / (Tr * FPWM) * 32767); // Diskretni IFOC koristi T/Tr
One_minus_Tpwm_over_Tr = (int)(1 - 1 / (Tr * FPWM)) * 32767; // i 1-T/Tr
Tpwm_over_TrPi_32b = (long)(1 / (Tr * FPWM * M_PI) * 0x7FFFFFFF); // i T/(TrPi)
SPEED_CONST=(int)(FREQ_SPEED_REG/((float)BASE_SPEED/60 *ENCODER_INC*4) *32767);
POSITION_CONST = (long)((double)0xFFFFFFFF / (ENCODER_INC * 4 - 1)); // QEI ..
SPEED_REG_PRESCAL = (int)((float)FPWM / FREQ_SPEED_REG); // svakih 1200 PWM
BAUD_RATE_GEN = (int)(FCY / (float)(16 * 19200) - 1); // clock serijske veze
}

```

Prvo se računa punjenje PTPER, tj. broj *clock* taktova koji čine jedan PWM period. Ovu vrednost koristi funkcija *SVPWM*() za punjenje PDC registara. Za izabrano $F_{CY} = 30\text{MHz}$ i $f_{PWM} = 12\text{kHz}$ ($1/12\text{kHz} \sim 83\mu\text{s}$) dobija se $PTPER = 30 \cdot 10^6 / (12 \cdot 10^3 \cdot 2) - 1 = 1249$. U računu je uzeto u obzir da je PWM modul podešen na trougaoni PWM nosilac, *up/down mode*.

Zatim se računa parametar vremenske konstante rotora koji se dalje koristi za račun konstanti *IFOC* kalkulatora, funkcija *IFOC_dq_angle_calc()*. Konstanta *Tpwm_over_Tr* je vrednost T_{PWM}/T_r u Q1.15. *One_minus_Tpwm_over_Tr* je $1 - T_{PWM}/T_r$ takođe u Q1.15. Konstanta *Tpwm_over_TrPi_32b* se koristi umesto vrednosti $T_{PWM}/(T_r \pi)$ u Q1.31.

Konstanta *SPEED_CONST* služi za procenu brzine obrtanja. Konstanta *POSITION_CONST* služi za pretvaranje vrednosti očitane iz *QEI* modula u relativan ugao u Q1.31 formatu. Konstanta *SPEED_REG_PRESCAL* određuje koliko će PWM prekida biti izvršeno između svake regulacije brzine obrtanja rotora.

Na kraju se frekvencija prenosa serijskih UART2 podataka podešava na 19200 b/s.

6.13.3.3. Funkcije za inicijalizaciju korišćenih programskih modula

Funkcija za start svih modula

U ovom poglavlju će biti opisane funkcije za inicijalizaciju perifernih modula koje koristi program. Nakon što su svi moduli inicijalizovani funkcija *StartModules()* ih sve startuje, tako što postavi bitove dozvole rada svih modula na jedan.

```

void StartModules() {
    ADCON1bits.ADON = 0x01; // Dozvoli rad ADC periferije
    PTCONbits.PTEN = 0x01; // Dozvoli rad PWM periferije
    U2MODEbits.UARTEN = 0x01; // Dozvoli rad UART periferije
}

```

Funkcija za inicijalizaciju GPIO - *IOPorts_Init()*

Funkcija *IOPorts_Init()* vrši inicijalizaciju kontrolnih registara ulazno-izlaznih portova. Kod dsPIC portovi su podeljeni u šest grupa (A, B, C, D, E i F). Svaka grupa ima registre za definisanje da li je port pin GPIO ulaz/izlaz ili povezan sa nekom periferijom.

```

void IOPorts_Init(){
    //ADC
    ADPCFGbits.PCFG0 = 0x00; // RB0 i RB6 - povezani sa ADC periferijom
    ADPCFGbits.PCFG6 = 0x00; // ADPCFG reg odlucuje - GPIO ili periferija ]
    TRISBbits.TRISB0 = 0x01; // RB0 i RB6 - input, TRISB da li su pinovi
    TRISBbits.TRISB6 = 0x01; // ulaz ili izlaz
    //QEI
    ADPCFGbits.PCFG4 = 0x01; // RB4 i RB5 povezani na QEI periferiju
    ADPCFGbits.PCFG5 = 0x01; //
    TRISBbits.TRISB4 = 0x01; // RB4 i RB5 digitalni ulazi
    TRISBbits.TRISB5 = 0x01; //
    //UART2
    TRISFbits.TRISF4 = 0x01; // RF4 i RF5 su ulazni pinovi
    TRISFbits.TRISF5 = 0x01; // za UART modul
}

```

Funkcija za inicijalizaciju UART modula - *UART_Init()*

UART modul služi za uspostavljanje asinhronne serijske veze sa perifernim uređajima, u ovom slučaju sa PC računarom. U programu se ovaj modul koristi da primi i privremeno sačuva 8-bitni podatak koji predstavlja referentnu brzinu obrtanja rotora. U okviru modula se nalazi 8-bitni prijemni registar (U2RSR) i niz od četiri 8-bitna registra za privremeno smeštanje primljenih podataka (U2RXREG). Modul se inicijalizuje za prenos 8-bitnih podataka bez bita parnosti. Prenosna frekvencija se podešava u U2BRG registru. Vrednost za upis u U2BRG registar se računa na osnovu $U2BRG = F_{CY}/(16 \cdot \text{Baud_Rate}) - 1$. Za izabranu *baudrate* od 19,2Kbps i $F_{CY} = 30\text{MHz}$, u U2BRG se upisuje broj 96.

```
void UART_Init() {
    U2MODEbits.PDSEL = 0x00;
    U2BRG = BAUD_RATE_GEN; }
```

Funkcija za inicijalizaciju PWM modula - *PWM_Init()*

PWM modulu (Motor Control PWM Module) je dodeljeno šest kontrolnih registara (PWMCON1, PWMCON2, DTCON1, DTCON2, FLTACON, FLTBCON), četiri registra za definisanje faktor ispune (PDC1, PDC2, PDC3, PDC4), registar za definisanje perioda PWM signala PTPER i vremenske baze (PTCON), i registar trenutnog stanja 16-bitnog brojača tj. PWM tajmera (PTMR).

```
void PWM_Init()
{
    PTCONbits.PTCKPS = 0x00;           // PWM clock prescaler = 1
    PTCONbits.PTMOD = 0x02;           // up/down mod tj. testera PWM nosioc
    PTPER = HALF_PWM_PERIOD;         // polovina jer se koriti PWM trougao !
    PTMR = 0x00;                     // reset PWM timera na vrednost nula
    PWMCON1bits.PMOD1 = 0x00;         // PWM grane 1,2,3 u komplemen. modu rada
    PWMCON1bits.PMOD2 = 0x00;         // Ovo znaci da upis u PDCx kontrolise i
    PWMCON1bits.PMOD3 = 0x00;         // gornji i donji prekidač iste grane
    PWMCON1bits.PEN1H = 0x01;         // Dozvoljava pojavu impulsa na PWM izlazu
    PWMCON1bits.PEN1L = 0x01;         // i tako za svih 6 PWM izlaza ka invertoru
    PWMCON1bits.PEN2H = 0x01;
    PWMCON1bits.PEN2L = 0x01;
    PWMCON1bits.PEN3H = 0x01;
    PWMCON1bits.PEN3L = 0x01;         // Komplementarni par PWM4 se ne koristi
    PWMCON2bits.OSYNC = 0x01;
    PDC1 = 0;                         // postavi sva tri faktora ispune na nulu
    PDC2 = 0;
    PDC3 = 0;
    IEC2bits.PWMIE = 0x01;           // dozvoli PWM prekid
    IFS2bits.PWMIF = 0x00;           // posto tek krecemo "ocisti" eventualni
}                                     // zahtev za prekid.
```

Funkcija za inicijalizaciju AD modula- *ADC_Init()*

Microchip dsPIC serija sadrži 10-bitni A/D konvertor koji konverziju vrši sukcesivnim aproksimacijama u trajanju od oko 2 μs . Postoji maksimum osam analognih ulaza (osam pinova mogu da se konfiguriše kao analogni ulazi) od kojih svi vode na analogni multiplexer koji je dalje povezan sa dva ili četiri S&H kola. Za podešavanje načina rada AD modula postoji šest kontrolnih registara. Po završetku konverzije rezultat se smešta u jedan od šesnaest 16-bitnih registara za prihvatanje i čuvanje rezultata (ADCBUF0 - ADCBUFF).

```
void ADC_Init()
{
    ADCON1bits.FORM = 0x03;           // tip formata rezultata, izabrano q1.15
    ADCON1bits.SSRC = 0x07;           // način početka konverzije, izabrano auto,
```

```

        ADCON1bits.SIMSAM = 0x01;    // sto znaci nakon samplovanja ide konverzija
        ADCON1bits.ASAM = 0x00;      // simultano S&H, oba sempluju istovremeno
        ADCON1bits.DONE = 0x00;      // Flag DONE (BUSY) zasad postavljen na nula,

        ADCON2bits.CHPS = 0x01;      // konvertuje kanale CH0 i CH1 , sa ta dva S&H
        // dve struje se citaju, treca racuna
        ADCON3bits.SAMC = 0x04;      // vreme uzimanja uzorak S&H kolom
        ADCON3bits.ADCS = 0x04;      // Podesavanje internog AD clock-a.
        // AD treba minumum 83,33ns za rad. Ovde je
        // preskaler 4 i dobije se 125ns sto je ok
        ADCHSbits.CH0SA = 0x06;      // Definise vezu izmedju pinova i S&H kola
        ADCHSbits.CH123SA = 0x00;
    }

```

Poslednje dve linije koda određuju vezu ulaznih pinova i S&H kola. Prvo se definiše da se na CH0 (S&H0) kolo dovodi signal sa pina RB6, na koji se dovodi signal sa *LEM* koji meri linijsku struju i_a . Zatim se CH1(S&H1) povezuje sa RB0 na kojem je signal proporcionalan linijskoj struji i_b . Time nakon konverzije, u registrima ADCBUF0 i ADCBUF1, imamo vrednosti i_a i i_b u Q1.15 formatu dok i_c računamo na osnovu njih u funkciji *ReadADC()*.

Funkcija za inicijalizaciju QEI modula - *QEI_Init()*

Kontroler poseduje QEI modul za brojanje impulsa sa inkrementalnog enkodera. U okviru modula nalaze se 16-bitni brojač, kontrolni registar, digitalni filter na svakom ulazu, registar maksimalne vrednosti brojača i logika za poređenje sadržaja brojača sa vrednošću registra. Signali sa enkodera (faze *A* i *B*, i *INDEX*) se dovode na tri pina koji su dodeljeni QEI modulu. Vrednost brojača se uveća svaki put po detekciji *UP*, i smanjuje po detekciji *DOWN* impulsa, poglavlje 6.7.2. Brojač se resetuje po dostizanju vrednosti u registru MAXCNT, ili po pristizanju *INDEX* impulsa, u zavisnosti kako je modul podešen da radi.

```

void QEI_Init()
{
    QEICONbits.UPDN = 0x01;          // smer brojanja pozitivan , A prednjaci B
    QEICONbits.QEIM = 0x07;          // Mod: 4*rezolucija i reset na MAXCNT
    MAXCNT = ENCODER_INC * 4 - 1;    // podesi MAXCNT na 4 puta broj impulsa/o
    POSCNT = 0x00;
}

```

6.13.3.4. Glavni program - *main()*

Po resetu mikrokontrolera poziva se asemblerski *startup* kôd koji priprema memorijske lokacije potrebne za rad višeg programskog jezika C. C program počinje da se izvršava pozivom *main* funkcije koja je opisana u ovom poglavlju. Na početku *main* funkcije se inicijalizuju svi periferni moduli pre nego što krenu da se koriste, zatim inicijalizuju se sve globalne konstante i sve promenljive potrebne za račun. Nakon inicijalizacije *main* program ulazi u beskonačnu petlju. U okviru beskonačne petlje rade se proračuni i radnje čije vreme izvršenja, kao i period između dva izvršenja nisu kritični. Funkcije glavne petlje se računaju samo u vremenu između dve PWM prekidne rutine i smatra se da je 25% i više preostalog vremena sasvim dovoljno. U ovom primeru programa u glavnoj petlji se vrši komunikacija sa računarom i za zadatu vrednost brzine uzima se 8-bitni podatak primljen sa udaljenog računara. Pošto je zadata brzina 16-bitni broj, bajt pristigao sa serijske veze se pomera za 8 bitova ulevo i tako se formira referentna brzina u normalizovanom opsegu ± 32767 .

```

int main()
{
    CONST_Init();                    // inicijalizacija konstanti i glob. promenljivih
    IOPorts_Init();                  // funkcije za init pojedinih perifernih modula
}

```

```

UART_Init();
PWM_Init();
ADC_Init();
QEI_Init();
StartModules();          // Periferni moduli su pripremljeni, startuj ih
while(1)
{
    if (U2STAbits.URXDA)          // ako je stigao podatak sa serijske veze
    {
        speed_ref = U2RXREG << 8; // uzmi ga i prebaci u referentnu brzinu
        U2STAbits.URXDA = 0;      // ukini signal (flag) da je stigao podatak
    }
}

```

6.13.3.5. Prekidna rutina PWM modula

Pošto su mehaničke vremenske konstante mnogo veće od PWM perioda nema potrebe za regulacijom brzine i upravljanje zadatim momentom AM u svakom PWM periodu. Zato ovaj primer programa tek nakon *SPEED_REG_PRESCAL* ulazaka u PWM prekid ulazi u deo za regulaciju brzine u kome prvo poziva estimator, a zatim i PI regulator brzine. Rezultat ovog dela programa je nova referenca q struje koja važi sledećih *SPEED_REG_PRESCAL* PWM perioda. U ovom primeru se zadata struja d ose ne menja, tj. nema slabljenja polja.

Ostatak PWM rutine je IVU i poziva se u svakom ulasku u prekid da bi najbrže moguće menjao vektor izlaznog napona, i u realnom vremenu pratio pomeraj vektora fluksa rotora. Ulazi su mu dq reference struje statora, a krajnji rezultat upis PWM vremena vođenja za novi PWM period. Pošto generisanje PWM signala ne sme da zastane, neophodno je da se celokupna rutina izvrši pre sledećeg PWM prekida (oko 83 μ s). Testovima je izmereno da ova rutina traje oko 1500 ciklusa (oko 50 μ s), tj. uzima oko 60% CPU vremena. Ovaj rezultat je dobijen sa svim funkcijama definisanim kao *inline* funkcije. Iako izgleda isto, *inline* funkcija u suštini nije funkcija. Programski prevodilac na mesto njenog poziva ugradi telo pozvane funkcije, umesto da je pozove. Time se uzima više memorije, ali izbegava rad sa argumentima, stekom, adresnim skokovima i skraćuje ukupno vreme izvršenja. *Inline* funkcije nisu deo standarda C jezika, ali Microchip prevodilac C30 ih podržava. U slučaju ostalih funkcija, prevodilac C30 samostalno odlučuje koje će prevesti kao standardne, a koje *inline*.

```

void __attribute__((__interrupt__)) _PWMInterrupt(void)
{
    interrupt_counter++;
    if (interrupt_counter > SPEED_REG_PRESCAL) // brzinska petlja svakih N
                                                // PWM perioda
    {
        interrupt_counter = 0;
        EstimateSpeed();                       // proracunaj brzinu
        PI_Speed_Regulator();                  // PI reg brzine zadaje Iq_ref
    }
    // sledi CRVSI i indirektna vektorska kontrola, svaki PWM se poziva
    ReadADC();                                // ocitaj Ia, Ib , izracunaj Ic
    Clark();                                  // Iabc -> Ial Ibata
    Park();                                   // Ial Ibata -> Id Iq
    IFOC_dq_angle_calc();                     // Id Iq theta_r -> theta_dq
    PI_Current_Regulator();                    // Idq_ref, Idq -> Vdref i Vqref
    InversePark();                             // Vdref i Vqref -> Valpha , Vbeta
    SVPWM();                                   // Valpha, Vbeta->PDC1,PDC2 i PDC3
    IFS2bits.PWMIF = 0x00;                    // ocisti zahtev za prekid
}

```

Zatim dolazimo do dela koda koji se izvršava svaki PWM period. To je čitanje struja, transformacija koordinata, čitanje ugla rotora i *IFOC* račun klizanja, regulacija struja u dq sistemu, transformacija dobijenih referentnih napona iz dq u $\alpha\beta$ domen i poziv SVPWM koja

računa nove faktore ispuše PDC1, PDC2 i PDC3. Na kraju, *IF* (*Interrupt Flag*) bit se postavlja na nulu, čime dajemo signal kontroleru prekida da smo prekid opslužili.

6.13.3.6. *IFOC* kalkulator klizanja i estimacija *dq* ugla – *IFOC_dq_angle_calc()*

Najvažnija i računarski najzahtevnija funkcija IVU programa je *IFOC* kalkulator klizanja. Njen izlaz je novi ugao *dq* sistema koji se računa na osnovu struja statora, ugla rotora i parametra vremenske konstante rotora. Od tačnosti njenog rezultata zavisi rad celokupnog indirektnog vektorskog upravljanja. Funkcija realizuje (6.163), (6.167) i (6.172).

Funkcija *IFOC* kalkulatora je vremenski zahtevna i zato je napisana u bržoj *fixed-point* aritmetici, pogl. 3.5. Radi preglednosti, potrebne linije koda za sve matematičke *fixed-point* operacije u C jeziku (tab. 3.5) su zamenjene leksičkim makroima. Program koristi makroe za množenje Q1.15xQ1.15, Q1.15xQ1.31 i deljenje Q1.15/Q1.15 (deljenje traje 29 *clock* ciklusa za razliku od *floating-point* C operatora *'/'* za čije izvršenje treba 478 ciklusa (oko 15μs!):

```
#define MULTIPLY_Q15(X,Y)          ((int)((long)(X)*(long)(Y) >> 15)
#define MULTIPLY_Q15_Q31(X,Y)     ((long long)(X)*(Y) >> 31)
#define DIVIDE_Q15(X, Y)          (__builtin_divsd((long)(X) << 15, (Y)))
```

Funkcija *IFOC_dq_angle_calc()* se poziva svaki PWM, nakon *Clark()* i *Park()* transformacija nad strujama i time koristi najnovije moguće odbirke struje za definisanje novog *dq* ugla i novih sinus / cosinus funkcija koje odgovaraju tom uglu.

```
inline void IFOC_dq_angle_calc()
{
    int Id_filt;
    long tmp32;
    int tmp;
    // filter id struje, (1.146), simulira dinamiku uspostavljanja fluksa
    Id_filt = MULTIPLY_Q15(Tpwm_over_Tr, Id_last);
    Id_filt += MULTIPLY_Q15(One_minus_Tpwm_over_Tr, Id_filt_last);
    Id_filt_last = Id_filt;
    // racun kolicnika Iq/Id , deo jednacine (1.167)
    tmp = Iq >> 4;
    if(tmp < Id_filt)    tmp = DIVIDE_Q15(tmp, Id_filt);
    else    tmp = 32767;
    // racun 32b inkrementa ugla klizanja
    tmp32 = MULTIPLY_Q15_Q31(tmp, Tpwm_over_TrPi_32b);
    tmp32 = tmp32 << 4;
    slip_angle += tmp32;                // Racun ugla klizanja, (6.167)
    rotor_pos = POSCNT * POSITION_CONST; // Racun merene pozicije rotora
    dq_angle = rotor_pos + slip_angle;   // Racun dq pozicije (1.172)
    dq_angle_q15 = dq_angle >> 16;      // 16-bitna verzija dq pozicije
    // Racun sin/cos logicki ne pripada ovde, ali se zbog vremena izvršenja racuna samo
    // jednom u PWM, i to ovde odmah nakon sto smo definisali novi dq ugao.
    // Ove iste sin i cosine koriste i Park struja i inverzni Park napona
    SIN_dqAngle = SinApprox(dq_angle_q15); // logicki ne pripadaju ovde ali
    COSINE_dqAngle = SinApprox(dq_angle_q15+16384); // cos(x)= sin(x+Pi)
}
```

Prvo se računa imenioc potreban za račun učestanosti klizanja, tj. realizuje jednačina (6.164) koja modeluje dinamiku uspostavljanja rotorskog fluksa. Ova jednačina je u suštini filter prvog reda sa vremenskom konstantom rotorskog kola. Prethodni izlaz filtra se pamti u globalnoj promenljivoj *Id_filt_last*. Filter koristi konstante *Tpwm_over_Tr* i *One_minus_Tpwm_over_Tr*, tj. T/T_r i $1-T/T_r$ u Q1.15 formatu. Ove konstante se mogu menjati i u toku rada pogona, u skladu sa promenom stvarne vremenske konstante rotora.

Zatim se vrši deljenje I_q/I_d_filt . Pošto je funkcija deljenja ograničena na rezultate manje od 1 (manje od 32767) pre deljenja se struja q ose подели sa 16 (pomeranje za 4 mesta udesno).

Time je obezbeđen tačan rezultat deljenja za q struju do 16 puta veću od d struje. Ukoliko je I_d (I_d_filt) veće od $I_q / 16$, rezultat deljenja se limituje na 32767.

Sledeće se vrši račun pomeraja ugla klizanja (6.170) za koji se koristi lokalna promenljiva $tmp32$. Pošto promena ugla usled klizanja za vreme jednog PWM perioda može biti izuzetno mala ugao klizanja se inkrementira i čuva u 32-bitnoj promenljivoj. Relativno mala konstanta $Tpwm_over_TrPi_32b$, zapisana u Q1.31 formatu, se množi sa rezultatom deljanja (tmp) u Q.15 formatu, primenom makroa `MULTIPLY_Q15_Q31`. Rezultat množenja se smešta u $tmp32$, koja se dalje množi sa brojem 16 da bi se poništilo skaliranje učinjeno pre deljenja dve struje. Izračunata 32-bitna vrednost promene ugla klizanja se zatim sabira sa prethodnim uglom klizanja i time smo dobili novi ugao klizanja.

Zatim se u promenljivu $rotor_pos$ učitava pozicija rotora iz registra `POSCNT` (QE1 modul) i pretvara u Q1.31 format (množi se sa `POSITION_CONST`). Vrednost konstante se računa u delu za inicijalizaciju na osnovu broja impulsa enkodera koji se koristi.

Nova pozicija dq koordinatnog sistema se dobija tako što se na očitano poziciju rotora ($rotor_pos$) dodaje izračunati ugao klizanja ($slip_angle$).

Konačno, za 16-bitnu vrednost ugla dq sistema se računaju odgovarajuće vrednosti sinus i cosinus funkcija. Račun sinus/cosinus funkcija logički ne pripada ovde već pripada u dve *Park* transformacije. Ipak, postavljen je ovde jer se u praksi vrši samo jednom u PWM periodu čime se značajno štedi na vremenu izvršenja a ne čini greška (pošto se dq ugao računa samo jednom u PWM periodu nema potrebe ni sinus/cosinus istog ugla računati više puta). Ako se sinus/cosinus račun vrši samo jednom, onda ga je najbolje vršiti u *IFOC* kalkulatoru, odmah nakon što je definisan novi ugao dq sistema. Iste sinus i cosinus vrednosti će se dalje višestruko koristiti tokom PWM perioda i zameniće vremenski zahtevne pozive funkcija. Na ovaj način je izbegnuto po dva poziva sinus i cosinus u *Park* i u inverznoj *Park* funkciji, sada imamo ukupno 2 umesto 8 poziva. Nadalje, pošto se *Park* transformacija struja vrši pre funkcije *IFOC_dq_angle_calc* () u njoj se koriste sinus/cosinus vrednosti računate u prethodnom PWM periodu što je i prirodno, jer su i struje merene u prethodnom PWM periodu. Inverzna *Park* napona se poziva nakon što je novi ugao definisan, koristi nove vrednosti sinus/cosinus koje važe za taj ugao i koje se slažu sa naponima aktivnim tek u sledećem PWM periodu. Ukoliko je odnos PWM i izlazne frekvencije dovoljno veliki (promena dq ugla tokom PWM perioda zanemarljiva), ovaj mehanizam je sasvim dovoljan. Za male odnose f_{PWM}/f_{out} , ultra velike brzine obrtanja, potrebne su dodatne intervencije.

Izvršenje *IFOC* funkcije je vremenski najzahtevnije: Ova funkcija ima jedno deljenje i sinus i cosinus funkcije na kraju, i za njeno izvršenje je potrebno oko $268 + 2 \cdot 200 = 668$ instrukcijskih ciklusa tj. oko $8,9\mu s + 2 \cdot 6,66\mu s = 22,6\mu s$. Treba imati na umu da je bi za PWM frekvenciju od 16kHz ova funkcija potrošila više od trećine raspoloživog vremena.

6.13.3.7. Ostale funkcije koje koristi C program

Funkcija *ReadADC()*

ADC modul vrši A/D konverziju signala sa mernih sondi i rezultat zapisuje u 16-bitne promenljive I_a i I_b . Na osnovu njih se računa vrednost promenljive I_c . Iskorišćena je mogućnost zapisa AD rezultata u Q1.15 formatu, izabrana prilikom inicijalizacije modula.

ReadADC() prvo startuje simultano odabiranje signala na oba S&H kola tako da imamo obe linijske struje iz istog vremenskog trenutka. Zatim program u dve *while* petlje čeka dok se simultano odabiranje (4 instrukcijska ciklusa) i dve konverzije koje počinju odmah nakon odabiranja ne završe. Prvo se konvertuje kanal `CH0` i rezultat se smešta u `ADCBUF0`, zatim kanal `CH1` i njegov rezultat se smešta u `ADCBUF1`. Konverzija svakog kanala traje 48 ciklusa. Kada je

konverzija oba kanala završena, bit `DONE` u okviru registra `ADCON1` postaje jednak jedinici i program izlazi iz druge *while* petlje. Rezultati se upisuju u promenljive *Ia* i *Ib* i na kraju izračuna *Ic*. Za izvršenje funkcije potreban je 81 ciklus.

```
void ReadADC() {
    ADCON1bits.SAMP = 0x01;
    while(ADCON1bits.SAMP);
    while(!ADCON1bits.DONE);
    ADCON1bits.DONE = 0x00;
    Ia = ADCBUF0;
    Ib = ADCBUF1;
    Ic = -(Ib + Ic);
}
```

Funkcija *Clark()*

Argumenti ove funkcije su fazne struje, promenljive *Ia*, *Ib* i *Ic*, Q1.15, definisane u *ReadADC()*. Izlazi su *I alpha* i *I beta*, Q1.15. Funkcija traje 29 instrukcijskih ciklusa.

```
inline void Clark() { // Clark transformacija, izraz 6.14
    I_alpha = Ia;
    I_beta = MULTIPLY_Q15(ONE_OVER_SQRT_3, Ib);
    I_beta -= MULTIPLY_Q15(ONE_OVER_SQRT_3, Ic)
}
```

Konstanta ONE_OVER_SQRT_3 čini da je *Clark* invarijantan po amplitudi i predstavlja broj $1/\sqrt{3}$ u Q1.15 formatu. Pošto za velike vrednosti struja može doći do prekoračenja u sabiranju dva Q1.15 broja prvo se sabirci množe sa $1/\sqrt{3}$, a tek onda sabiraju.

Funkcije *Park()* i *InversePark()*

Argumenti *Park* funkcije su *I_alpa* i *I_beta* izračunati u *Clark()* i *dq* ugao izračunat u *IFOC_dq_angle_calc()*, svi važeći za prethodni PWM period. Izlaz funkcije su statorske struje u *dq* rotirajućem sistemu, *Id* i *Iq*, u Q1.15 formatu.

```

inline void Park(){          // Park transformacija, izraz 6.17
    int temp_1,temp_2;
    Id_last = Id;           // pamti prethodnu Id za inkrem. reg. i za filter
    temp_1 = MULTIPLY_Q15(I_alpha, COSINE_dqAngle); // Struje I_alpha i I_beta
    temp_2 = MULTIPLY_Q15(I_beta, SIN_dqAngle);      // iz Clark(), ugao iz
    Id = temp_1 + temp_2;           // IFOC_dq_angle_calc()
    Iq_last = Iq;                  // pamti se prethodna vrednost Iq inkrem. regulator
    temp_1 = MULTIPLY_Q15(I_alpha, SIN_dqAngle);
    temp_2 = MULTIPLY_Q15(I_beta, COSINE_dqAngle);
    Iq = -temp_1 + temp_2;          //Id i Iq spremne za strujne
}                                   //regulatore

```

InversePark() se poziva nakon što je izračunat novi dq ugao i nove dq komande napona (U_d i U_q), tako da računa nove U_{alpha} i U_{beta} koji važe za sledeći PWM period.

```

inline void InversePark(){ // Inverzna. Park transformacija, izraz 6.20
int temp_1,temp_2;
temp_1 = MULTIPLY_Q15(Ud, COSINE_dqAngle); // Ud i Uq na izlazu PI
temp_2 = MULTIPLY_Q15(Uq, SIN_dqAngle); // regs struje
U_alpha = temp_1 - temp_2; // Sine i Cosine dobijeni u
temp_1 = MULTIPLY_Q15(Ud, SIN_dqAngle); // IFOC_dq_angle_calc()
temp_2 = MULTIPLY_Q15(Uq, COSINE_dqAngle);
U_beta = temp_1 + temp_2; // Novi U_alpha i U_beta
} // spremni za SVPWM()

```

Funkcija `PI_Current_Regulator()`

U programu postoje dva identična ali nezavisna PI regulatora struje, jedan za d i jedan za q osu, pogl. 6.8.2. Ulazne promenljive regulatora su Id , Id_last , Iq , Iq_last , sve u Q1.15 formatu. Funkcija koristi konstante Kp_i , Ki_i i merenu ili nominalnu vrednost napona jednosmernog kola. Izlazi funkcije su komponente referentnog naponskog vektora Ud i Uq . Za izvršenje funkcije potrebno je 158 instrukcijska ciklusa.

Vrednosti Ud i Uq treba ograničiti tako da amplituda naponskog vektora ne bude veći od $u_{DC}/\sqrt{3}$. Iz tog razloga, $u_{sd}^{lim} = u_{DC}/\sqrt{3}$, a $u_{sq}^{lim} = \sqrt{u_{DC}^2/3 - u_{sd}^2}$. Dati pojednostavljeni primer IVU programa ne predviđa merenje napona DC kola, tako da dole priložen listing funkcije ne koristi merenu, nego konstantnu nominalnu vrednost DC napona, UDC_NOM .

Standardna *floating-point* funkcija `sqrt()` iz *math.h* biblioteke je slično kao i `sin()` izuzetno vremenski zahtevna, i zbog toga je kvadratni koren realizovan u vidu *lookup* tabele. Funkcija kvadratnog korena je u granicama $[0, 1]$ aproksimirana *lookup* tabelom sa 256 vrednosti. Za njen poziv se koristi makro `SQRT(X)`.

Oba PI regulatora su u inkrementalnoj formi sa proporcionalnim dejstvom izmeštenim u lokalnu povratnu spregu. Zato se umesto $Kp(err - err_old)$ koristi $Kp(Id_last - Id)$.

U regulatorima nisu realizovani *feed-forward* članovi, tako da uticaj sprege (*cross-coupling*) između osa nije eliminisan. Za pogon koji radi na velikim brzinama uticaj sprege se uvećava i potrebno je uvesti potrebne članove u jednačine regulatora, pogl. 6.8.2.

```
#define SQRT(X)          SqrtApprox[(X) >> 7]
inline void PI_Current_Regulator()
{
    int tmp,tmp1;
    tmp = MULTIPLY_Q15(Kp_i, (Id_last - Id));      // izmesteno P dejstvo
    tmp += MULTIPLY_Q15(Ki_i, Id_ref - Id);        // I dejstvo
    Ud += tmp;
    tmp1 = MULTIPLY_Q15(UDC_NOM, ONE_OVER_SQRT_3); // Udlim= udc/sqrt(3)
    if (Ud >= tmp1) Ud = tmp1;
    if (Ud <= -tmp1) Ud = -tmp1;
    tmp = MULTIPLY_Q15(Kp_i, (Iq_last - Iq));      // izmesteno P dejstvo
    tmp += MULTIPLY_Q15(Ki_i, Iq_ref - Iq);        // I dejstvo
    Uq += tmp;
    tmp = MULTIPLY_Q15(Ud, Ud);
    tmp1 = MULTIPLY_Q15(tmp1, tmp1);
    UQ_LIMIT = SQRT(tmp1 - tmp);                    // Uqlim=sqrt(udc^2/3 -ud^2)
    if (Uq >= UQ_LIMIT)    Uq = UQ_LIMIT;
    if (Uq <= -UQ_LIMIT)   Uq = -UQ_LIMIT;
}
```

Funkcija `EstimateSpeed()`

Funkcija vrši procenu brzine obrtanja rotora na osnovu broja nadošlih impulsa, tj. razlike u poziciji inkrementalnog enkodera detektovanoj u toku perioda merenja brzine (10ms). Konstanta kojom se množi broj inkremenata u programu je $SPEED_CONST$, i njena vrednost je izračunata u delu za inicijalizaciju. Pošto brojač QEI modula radi po modulu jednakom broju impulsa sa enkodera, račun promene pozicije rotora je nešto složeniji nego u slučaju brojača po modulu 32767. Račun promene zahteva proveru da li je brojač QEI modula između dva perioda čitanja dostigao maksimalnu vrednost. Ukoliko se rotor obrće u pozitivnom smeru, maksimum se desio ako je nova vrednost manja od prethodno pročitane. Slično je za negativan smer. Ako se utvrdi da je došlo do prekoračenja, razlika dobijena oduzimanjem vrednosti brojača u dva perioda se koriguje, pogledati program. Procenjenu vrednost brzine dalje koristi PI regulator brzine koji se sledeći poziva.

```

inline void EstimateSpeed()
{
    int tmp;
    tmp = POSCNT - POSCNT_last;
    if (abs(tmp) > 2*SPEED_CONST)
    {
        if (POSCNT < POSCNT_last)
            tmp += ENCODER_INC * 4;
        else if (POSCNT > POSCNT_last)
            tmp -= ENCODER_INC * 4;
    }
    speed = (int)(SPEED_CONST * tmp);
    POSCNT_last = POSCNT;
}

```

Prvo se izračuna razlika između pozicije enkodera iz prethodnog perioda, koja je smeštena u globalnoj promenljivoj *POSCNT_last*, i trenutne pozicije koja se nalazi u *POSCNT* registru QE1 modula. Zatim se vrši provera prekoračenja maksimalne vrednosti brojača QE1 modula. Pri tome se smatra da se rotor nikad neće obrtati većom brzinom od dvostruke maksimalne. Ukoliko je došlo do prekoračenja, apsolutna vrednost promenljive *tmp* je veća od $2 \cdot \text{SPEED_CONST}$, i program ispravlja razliku u poziciji tako što na vrednost u promenljivoj *tmp* doda ili oduzme broj impulsa enkodera po jednom obrtaju. Zatim se određuje smer obrtanja. Konačno, računa se procena brzine obrtanja rotora. Brzina se računa tako što se razlika u poziciji množi sa konstantom *SPEED_CONST*. Konstanta *SPEED_CONST* je izračunata tako da se za maksimalnu *BASE_SPEED* dobije vrednost promenljive *speed* od 32767. Za izvršenje funkcije je potrebno 62 ciklusa.

Funkcija *PI_Speed_Regulator()*

Uloga funkcije je da izračuna nove referentne vrednosti *dq* komponenti statrorske struje kako bi se brzina održala na željenoj vrednosti. PI regulator je urađen u istoj formi kao strujni, dakle u inkrementalnoj formi sa pozicionim dejstvom izmeštenim u povratnu spregu. Pošto nije predviđeno da se motor obrće brzinom većom od nominalne, vrednost komponente *Id_ref* se ne izračunava već joj se dodeljuje nominalna vrednost. Ulazne promenljive funkcije *PI_Speed_Regulator()* su nova i stara brzina (*speed*, *speed_last*), izlazne su zadate *dq* struje *Iq_ref* i *Id_ref*. Za izvršenje funkcije je potrebno 42 ciklusa.

```

inline void PI_Speed_Regulator()
{
    int tmp;
    tmp = MULTIPLY_Q15(-Kp_speed, (speed - speed_last)); // Kp samo u lok. grani
    tmp += MULTIPLY_Q15(Ki_speed, (speed_ref - speed));
    Iq_ref += tmp;
    Id_ref = ID_NOM;
    speed_last = speed;
}

```

Funkcija *SVPWM()*

Funkcija *SVPWM()* se poziva poslednja u PWM prekidu. Njena uloga je da na osnovu $\alpha\beta$ komponenti vektora referentnog napona upiše nove vrednosti u odgovarajuće registre faktora ispunje PDC1, PDC2 i PDC3. Ona interno određuje sektor u kome je vektor, računa potrebna vremena aktivnih vektora koji graniče taj sektor i konačno računa faktore ispunje. Ulazne promenljive funkcije *SVPWM()* su *U_alpha*, *U_beta*. Globalne promenljive koje koristi su *sector*, *big*, *middle* i *small*. Za izvršenje funkcije na dsPIC4011 potrebno je 247 ciklusa, tj. 8.2 μs na 30MHz. Detaljnije o realizaciji funkcije SVPWM u pogl. 5.3.